PATENT ABSTRACTS OF JAPAN

(11)Publication number : 07-191892
(43)Date of publication of application : 28.07.1995

(51)Int.Cl. G06F 12/00
G06F 12/00
G11C 16/06

(21)Application number : 05-013980
(22)Date of filing : 29.01.1993

(71)Applicant : MICROSOFT CORP
(72)Inventor : KRUEGER WILLIAM J
RAJAGOPALAN SRIRAM

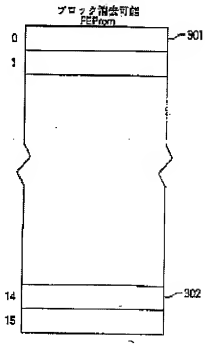
(30)Priority
Priority number : 92 828763 Priority date : 29.01.1992 Priority country : US

(54) METHOD FOR MANAGING FILE SYSTEM BY USING FLASH ERASABLE PROGRAMMABLE READ ONLY MEMORY AND SYSTEM THEREFOR

(57)Abstract:

PURPOSE: To provide a method and system for managing a file stored in block erasable and flash erasable programmable read only memory.

CONSTITUTION: A manager for managing a file is provided with a block assigning routine for selecting a block 302 at a memory position for storing data, and each block 302 is provided with an assignment table and a data area divided into data areas, and the assignment table is provided with an entry corresponding to the data area. Moreover, the manager is provided with a data area assigning routine for selecting the data area in the data area for the selected block, selecting the entry of the assignment table so as to be made correspond to the data area, and setting the entry selected so as to be made correspond to the selected data area and the assigned state.



(19)日本国特許庁（ＪＰ）　(12) 公開特許公報(A)　(11)特許出願公開番号

特開平7－191892

(43)公開日　平成７年(1995)７月28日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ	技術表示箇所
Ｇ０６Ｆ 12/00	５２０ Ｊ	8944－5Ｂ		
	５０１ Ｈ	8944－5Ｂ		
Ｇ１１Ｃ 16/06				
			Ｇ１１Ｃ 17/00	５３０ Ｂ

審査請求　未請求　請求項の数41　ＯＬ　（全 34 頁）

(21)出願番号　特願平5－13980

(22)出願日　平成５年(1993)１月29日

(31)優先権主張番号　０７／８２８７６３

(32)優先日　1992年１月29日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　391055933
マイクロソフト　コーポレイション
ＭＩＣＲＯＳＯＦＴ　ＣＯＲＰＯＲＡＴＩＯＮ
アメリカ合衆国　ワシントン州　98052－6399　レッドモンド　ワン　マイクロソフト　ウェイ　（番地なし）

(72)発明者　ウィリアム　ジェイ　クルーガー
アメリカ合衆国　ワシントン州　98053　レッドモンド　トゥーハンドレッドアンドエイス　アベニュー　ノースイースト　6320

(74)代理人　弁理士　中村　稔　（外６名）

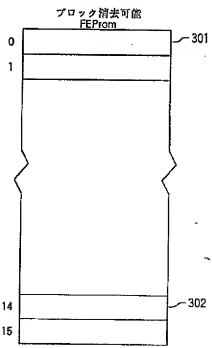
最終頁に続く

(54)【発明の名称】　フラッシュ消去可能なプログラマブル・リードオンリメモリを用いてファイルシステムをマネージする方法及びシステム

(57)【要約】　（修正有）

【目的】　ブロック消去可能で且つフラッシュ消去可能なプログラマブル・リードオンリメモリに記憶されたファイルをマネージする方法及びシステムを提供する。

【構成】　ファイルをマネージするためのマネージャは、データを記憶するメモリ位置のブロックを選択するためのブロック割り当てルーチンを備えており、各ブロックは、割り当てテーブルと、データエリアに分割されたデータ領域とを有し、割り当てテーブルはデータエリアに対応するエントリを有している。更に、上記マネージャは、上記選択されたブロックに対してデータ領域内のデータエリアを選択し、それに対応するように割り当てテーブルのエントリを選択し、そして選択されたデータエリア及び割り当てられた状態に対応するように上記選択されたエントリをセットするためのデータエリア割り当てルーチンを備えている。



1

[特許請求の範囲]

【請求項1】　コンピュータメモリをメモリ位置のブロックに分割し、各ブロックは、割り当てテーブルと、データエリアに分割されたデータ領域とを有するものであり、上記割り当てテーブルは上記領域のデータエリアに対応するエントリを有し、そしてデータを記憶するブロックを選択するためのブロック割り当てルーチンと、

データを記憶する上記選択されたブロックに対しデータ領域内のデータエリアを選択し、その選択されたデータエリアに対応するように割り当てテーブルのエントリを選択し、そして上記選択されたデータエリア及び割り当てられた状態に対応するように上記選択された割り当てテーブルエントリをセットするためのデータエリア割り当てルーチンと、

上記選択されたデータエリアにデータを記憶するための記憶ルーチンとを備えたことを特徴とするコンピュータメモリ用のマネージャ。

【請求項2】　割り当てられた状態にある割り当てテーブルエントリを割り当て解除状態にセットするためのデータエリア割り当て解除ルーチンと、

その割り当て解除された状態にある割り当てテーブルエントリに対応するデータエリアをリクレイミングするためのブロックリクレイマルーチンとを更に備えた請求項1に記載のコンピュータメモリ用のマネージャ。

【請求項3】　各ブロックがヘッダ情報を有し、これらヘッダ及び割り当てテーブルから情報を収集しそしてその収集した情報をメモリキャッシュに記憶するための初期化ルーチンを更に備えた請求項1に記載のコンピュータメモリ用のマネージャ。

【請求項4】　1つ以上の論理ブロックを有する一回書き込み多数回読み取りのメモリデバイスにおいて、

データを記憶するためのエリアに論理的に分割される各ブロック内のデータ領域と、

各ブロック内に記憶されるブロック構造体であって、ヘッダと割り当てテーブルとを有していて、ヘッダはブロックに特定の情報を含み、割り当てテーブルはデータ領域のエリアに対応するエントリを有していると共に、その対応するデータ領域のエリアに関連した情報を含んでいるようなブロック構造体とを備えたことを特徴とするメモリデバイス。

【請求項5】　上記ヘッダは、論理ブロック番号と、ブロック状態と、ブロック消去カウントとを含む請求項4に記載のメモリデバイス。

【請求項6】　割り当てテーブルエントリは、対応するデータ領域のエリアを指すポインタと、対応するデータ領域のエリアの長さを指示するレングスと、割り当てテーブルエントリのステータスとを含んでいる請求項4に記載のメモリデバイス。

【請求項7】　上記メモリデバイスは、ブロック消去可能なプログラマブル・リードオンリメモリである請求項4に記載のメモリデバイス。

2

【請求項8】　複数のブロックを備えたブロック消去可能なプログラマブル・リードオンリメモリにおいてメモリをマネージする方法が、

各ブロックにブロックヘッダ情報を記憶し、

各ブロックに割り当てテーブルを記憶し、そして各ブロックのデータ記憶エリアにデータを記憶する、という段階を備えたことを特徴とする方法。

【請求項9】　上記ブロックヘッダ情報は、論理ブロック番号と、ブロック状態と、ブロック消去カウントとを含んでいる請求項8に記載の方法。

【請求項10】　上記割り当てテーブルはデータ記憶エリアの一部分に対応するエントリを含み、これらのエントリは、データ記憶エリアの対応する部分を指すポインタと、データ記憶エリアの対応部分の長さを指示するレングスと、エントリの状態を指示するステータスとを含む請求項8に記載の方法。

【請求項11】　ブロックを備えたブロック消去可能なプログラマブル・リードオンリメモリにおいて割り当て解除されたスペースをリクレイミングする方法が、

リクレイミングされるべきブロックにおいて割り当て解除された又は割り当てられたものとしてデータ領域を識別し、

スペアブロックを消去し、そしてリクレイミングされるべきブロックからスペアブロックへ割り当てられたデータ領域をコピーし、それにより、割り当て解除されたデータ領域に対応するメモリエリアを割り当てのためにリクレイミングする、という段階を備えたことを特徴とする方法。

【請求項12】　割り当てられたデータ領域はスペアブロック内の隣接するメモリ位置にコピーされる請求項11に記載の方法。

【請求項13】　コンピュータメモリデバイス内のデータ領域をアドレスする方法において、上記メモリはブロックに分割され、そして各ブロックは物理的なブロック番号を有しており、上記方法は、

各ブロックに割り当てテーブルを記憶し、この割り当てテーブルは、ブロック内のデータ領域のオフセットを指示するエントリと、エントリインデックスとを有しており、

更に、各ブロックに論理ブロック番号を記憶し、

この論理ブロック番号と、割り当てテーブルのエントリインデックスとによってデータ領域を識別し、そして上記論理ブロック番号と、割り当てテーブルのエントリインデックスとに基づいてその識別されたデータ領域に対するアドレスを発生する、という段階を備えたことを特徴とする方法。

【請求項14】　アドレスを発生する上記段階は、

上記論理ブロック番号から物理的なブロック番号を決定し、各ブロックは対応するスタートアドレスを有し、

上記割り当てテーブルのエントリインデックスによって指示された上記決定された物理的なブロック番号において上記割り当てテーブルのエントリからオフセットを検索し、そしてその検索されたオフセットを上記決定された物理的なブロック番号と共にブロックのスタートアドレスに追加して、識別されたデータ領域のアドレスを発生するという段階を含む請求項１３に記載の方法。

【請求項１５】　コンピュータメモリデバイスは、ブロック消去可能なプログラマブル・リードオンリメモリである請求項１３又は１４に記載の方法。

【請求項１６】　コンピュータメモリデバイスのブロックを識別する方法において、各ブロックは物理的なブロック番号を有しており、上記方法は、

各ブロックに論理ブロック番号を配憶し、

各論理ブロック番号から、その論理ブロック番号が配憶された物理的なブロック番号へ至るマップを発生し、

論理ブロック番号を受け取り、そしてその受け取った論理ブロック番号を、上記発生されたマップを用いて物理的なブロック番号に変換する、という段階を備えたことを特徴とする方法。

【請求項１７】　ブロックをリクレイミングするときに、リクレイミングされるべきブロックからリクレイミングされたブロックへ論理ブロック番号をコピーする段階を備えた請求項１６に記載の方法。

【請求項１８】　コンピュータメモリデバイスは、ブロック消去可能なプログラマブル・リードオンリメモリである請求項１６又は１７に記載の方法。

【請求項１９】　複数のブロックを有するブロック消去可能なプログラマブル・リードオンリメモリにおいてブロックの消去カウントを維持する方法が、

ブロックを消去した回数を指示する消去カウントを各ブロックに記憶し、

第１ブロックを第２ブロックにコピーするときに、第２ブロックの消去カウントを保持し、そしてブロックを消去するときに、その消去の前に消去カウントを増加しそしてその増加した消去カウントを消去したブロックに記憶する、という段階を備えたことを特徴とする方法。

【請求項２０】　データを記憶するためのブロック消去可能なプログラマブル・リードオンリメモリにおいてブロックを割り当てる方法が、

ブロックを消去した回数を示す消去カウントを各ブロックごとに維持し、

データを記憶するに充分なスペースを有するブロックを選択し、そしてデータを記憶するためにその消去カウントに基づいて上記選択されたブロックを識別し、ブロックの割り当てを行う、という段階を備えたことを特徴とする方法。

【請求項２１】　ブロック消去可能なプログラマブル・リードオンリメモリにおいてブロックの消去を均等化する方法が、

消去された第１ブロックを識別し、

第１ブロックよりも少ない回数で消去された第２ブロックを識別し、そして第１ブロックのデータを第２ブロックのデータとスワッピングする、という段階を備えたことを特徴とする方法。

【請求項２２】　複数のディレクトリエントリを有するハイアラーキ式のディレクトリ構造体に基づいてファイルを編成するためのコンピュータファイル記憶システムにおいて、

ディクトリエントリを記憶するメモリを有するコンピュータと、

ディレクトリエントリを記憶するメモリの一部分を割り当てる手段であって、ディレクトリエントリは、一次ポインタと、二次ポインタと、兄弟ポインタとを有し、これらのポインタが最初に所定値にセットされているような割り当て手段と、

ディレクトリエントリの上記兄弟ポインタを、ディレクトリ構造体の同じレベルにある別のディレクトリエントリを指すようにセットして、兄弟ディレクトリエントリのリンクされたリストを形成するための手段と、

ディレクトリエントリの上記一次ポインタを、ディレクトリ構造体の次に低いレベルにある別のディレクトリエントリを指すようにセットする手段と、

取って代えられるディレクトリエントリの上記二次ポインタを、取って代わるディレクトリエントリを指すようにセットする手段とを具備し、この取って代わるディレクトリエントリは、取って代えられるディレクトリエントリの更新されたデータを含むことを特徴とするコンピュータファイル記憶システム。

【請求項２３】　上記取って代わるディレクトリエントリの一次ポインタを上記取って代えられるディレクトリエントリの一次ポインタに等しくセットするための手段と、

上記取って代わるディレクトリエントリの兄弟ポインタを上記取って代えられるディレクトリエントリの兄弟ポインタに等しくセットするための手段とを更に備えた請求項２２に記載のコンピュータファイル記憶システム。

【請求項２４】　ファイルエントリを記憶するためのメモリの一部分を割り当てる手段を更に備え、ファイルエントリは、関連ファイルに関する情報を含むものであり、そしてディレクトリエントリの一次ポインタをその割り当てられたファイルエントリを指すようにセットする手段を更に備えた請求項２２に記載のコンピュータファイル記憶システム。

【請求項２５】　上記の割り当てられたファイルエントリは、一次ポインタと、二次ポインタと、最初に所定値にセットされる兄弟ポインタとを含み、

ファイルエントリの上記兄弟ポインタを、別のファイル又はディクトリエントリを指すようにセットして、ファイル又はディレクトリエントリのリンクされたリスト

を形成するための手段と，

取って代えられるファイルエントリの上記二次ポインタを、取って代わるファイルエントリを指すようにセットする手段とを具備し、この取って代わるファイルエントリは、取って代えられるファイルエントリの更新されたデータを含む請求項２４に記載のコンピュータファイル記憶システム。

【請求項２６】　上記取って代わるファイルエントリの一次ポインタを上記取って代えられるファイルエントリの一次ポインタに等しくセットするための手段と、

上記取って代わるファイルエントリの兄弟ポインタを上記取って代えられるファイルエントリの兄弟ポインタに等しくセットするための手段とを更に備えた請求項２５に記載のコンピュータファイル記憶システム。

【請求項２７】　ファイル情報エントリを記憶するためのメモリの一部分を割り当てる手段を更に備え、ファイル情報エントリは、ファイルエクステントに関する情報を含むものであり、そしてファイルエントリの一次ポインタをその割り当てられたファイル情報エントリを指すようにセットする手段を更に備えた請求項２４に記載のコンピュータファイル記憶システム。

【請求項２８】　上記ファイル情報エントリは、最初に所定値にセットされる一次ポインタ及び二次ポインタを含み，

ファイル情報エントリの上記一次ポインタを、その同じファイルに関連した別のファイル情報エントリを指すようにセットして、その同じファイルに対するファイル情報エントリのリンクされたリストを形成するための手段と，

取って代えられるファイル情報エントリの上記二次ポインタを、取って代わるファイル情報エントリを指すようにセットする手段とを具備し、この取って代わるファイル情報エントリは、取って代えられるファイル情報エントリの更新されたデータを含む請求項２７に記載のコンピュータファイル記憶システム。

【請求項２９】　上記取って代わるファイル情報エントリの一次ポインタを上記取って代えられるファイル情報エントリの一次ポインタに等しくセットする手段を更に備えた請求項２８に記載のコンピュータファイル記憶システム。

【請求項３０】　ファイルを記憶するためのコンピュータファイル記憶システムにおいて、

ファイルを記憶するメモリを有するコンピュータと，

ファイルエントリを記憶するメモリの一部分を割り当てる手段であって、ファイルエントリは、それに関連したファイルに関する情報を含むと共に、一次ポインタと、二次ポインタと、最初に所定値にセットされる兄弟ポインタとを有しているような割り当て手段と，

ファイルエントリの上記兄弟ポインタを、別のファイルエントリを指すようにセットして、ファイルエントリのリンクされたリストを形成する手段と，

取って代えられるファイルエントリの上記二次ポインタを、取って代わるファイルエントリを指すようにセットする手段とを具備し、この取って代わるファイルエントリは、取って代えられるファイルエントリの更新されたデータを含むことを特徴とするコンピュータファイル記憶システム。

【請求項３１】　上記取って代わるファイルエントリの一次ポインタを上記取って代えられるファイルエントリの一次ポインタに等しくセットするための手段と，

上記取って代わるディレクトリエントリの兄弟ポインタを上記取って代えられるディレクトリエントリの兄弟ポインタに等しくセットするための手段とを更に備えた請求項３０に記載のコンピュータファイル記憶システム。

【請求項３２】　上記ファイル情報エントリは、最初に所定値にセットされる一次ポインタ及び二次ポインタを含み、

ファイル情報エントリを記憶するためのメモリの一部分を割り当てる手段を更に備え、ファイル情報エントリは、ファイルのファイルエクステントに関する情報を含むものであり、

更に、ファイルエントリの一次ポインタをその割り当てられたファイル情報エントリを指すようにセットする手段と，

ファイル情報エントリの上記一次ポインタを、その同じファイルに関連した別のファイル情報エントリを指すようにセットして、その同じファイルに対するファイル情報エントリのリンクされたリストを形成するための手段と，

取って代えられるファイル情報エントリの上記二次ポインタを、取って代わるファイル情報エントリを指すようにセットする手段とを具備し、この取って代わるファイル情報エントリは、取って代えられるファイル情報エントリの更新されたデータを含む請求項３０に記載のコンピュータファイル記憶システム。

【請求項３３】　上記取って代わるファイル情報エントリの一次ポインタを上記取って代えられるファイル情報エントリの一次ポインタに等しくセットする手段を更に備えた請求項３２に記載のコンピュータファイル記憶システム。

【請求項３４】　コンピュータメモリデバイスに記憶されたデータを新たなデータで更新する方法であって、上記メモリデバイスはデータの記録を含んでおり、各記録は最初に所定値を有する二次ポインタを備えており、上記方法は、

更新されるべきデータを含む記録を探索し，

新たなデータを含むように記録を割り当て，

その割り当てられた記録に新たなデータを書き込み、そして上記探索された記録における二次ポインタを上記割り当てられた記録を指すようにセットし、その割り当て

られた記録の新たなデータが上記探索された記録におけるデータの更新であることを指示するようにする、という段階を備えたことを特徴とする方法。

【請求項３５】　二次ポインタをセットする上記段階は、二次ポインタが所定値から変化したことを示すように上記探索された記録にフラグをセットすることを含む請求項３４に記載の方法。

【請求項３６】　１つ以上の論理ブロックを有する一回書き込み多数回読み取りのメモリデバイスを備えたコンピュータシステム内でエラーを回復するための方法において、

（ａ）各ブロックごとにデータ領域を画成し、データ領域は、データを記憶するためのデータ領域エリアに論理的に分割され、

（ｂ）各ブロックごとにブロック構造体を画成し、ブロック構造体はヘッダと割り当てテーブルとを有し、ヘッダはブロックに特定の情報を含み、割り当てテーブルは、データ領域エリアに対応するエントリを有すると共に、対応するデータ領域エリアに関連した情報を含んでおり、

（ｃ）割り当てテーブルエントリを割り当て、

（ｄ）データを記憶するためのデータ領域エリアを割り当て、

（ｅ）その割り当てられたデータ領域エリアに関連したデータをその割り当てられた割り当てテーブルエントリに書き込み、

（ｆ）その割り当てられたデータ領域エリアにデータを書き込み、

（ｇ）データ領域エリアに書き込みをしながらエラーを検出し、そして

（ｈ）エラーを検出した際には、割り当てテーブルエントリを割り当て解除状態にセットし、そしてデータ領域エリアにデータを書き込む間にエラーが検出されなくなるまで上記段階（ｃ）、（ｄ）、（ｅ）及び（ｆ）を繰り返すことを特徴とする方法。

【請求項３７】　１つ以上の論理ブロックを有する一回書き込み多数回読み取りのメモリデバイスを備えたコンピュータシステム内でエラーを回復するための方法において、

（ａ）各ブロックごとにデータ領域を画成し、データ領域は、データを記憶するためのデータ領域エリアに論理的に分割され、

（ｂ）各ブロックごとにブロック構造体を画成し、ブロック構造体はヘッダと割り当てテーブルとを有し、ヘッダはブロックに特定の情報を含み、割り当てテーブルは、データ領域エリアに対応するエントリを有すると共に、対応するデータ領域エリアに関連した情報を含んでおり、

（ｃ）割り当てテーブルエントリを割り当て、

（ｄ）データを記憶するためのデータ領域エリアを割り当て、

（ｅ）その割り当てられたデータ領域エリアに関連したデータをその割り当てられた割り当てテーブルエントリに書き込み、

（ｆ）その割り当てられたデータ領域エリアにデータを書き込み、

（ｇ）その割り当てられた割り当てテーブルエントリを割り当て状態にセットし、

（ｈ）その割り当てられた割り当てテーブルエントリを割り当て状態にセットしながらエラーを検出し、そして

（ｉ）エラーを検出した際には、割り当てテーブルエントリをゼロ状態にセットし、そして割り当てられた割り当てテーブルエントリにデータを書き込む間にエラーが検出されなくなるまで上記段階（ｃ）、（ｅ）及び（ｇ）を繰り返すことを特徴とする方法。

【請求項３８】　消去可能な１つ以上の論理ブロックを有する一回書き込み多数回読み取りのメモリデバイスを備えたコンピュータシステム内でエラーを回復するための方法において、

上記ブロックに対してブロックの状態を含むブロックヘッダデータ構造体を画成し、

そのブロックヘッダデータ構造体にデータを書き込み、そしてそのブロックヘッダデータ構造体にデータを書き込む間にエラーが検出された際に、そのブロックを消去状態のための待ち行列にセットし、これにより、そのブロックは、それが消去されるまでデータの記憶に使用されないようにすることを特徴とする方法。

【請求項３９】　消去可能な１つ以上の論理ブロックを有する一回書き込み多数回読み取りのメモリデバイスを備えたコンピュータシステム内でエラーを回復するための方法において、

上記ブロックに対してブロックの状態を含むブロックヘッダデータ構造体を画成し、

上記ブロックを消去し、そしてそのブロックの消去中にエラーが検出された際に、そのブロックをリタイア状態にセットし、これにより、そのブロックは、データの記憶に使用されないようにすることを特徴とする方法。

【請求項４０】　コンピュータメモリマネージャにおいて指定のメモリデバイスがブロック消去可能でないことを指定する方法であって、上記コンピュータメモリマネージャはブロック消去可能な一回書き込み多数回読み取りのメモリデバイスをマネージするものであり、上記メモリマネージャはスペアブロックのカウントを維持し、更に、上記メモリマネージャは、データをスペアブロックにコピーし、指定されたブロックを消去しそしてその指定されたブロックをスペアブロックとなるようセットすることによってその指定されたブロック内のデータを圧縮するものであり、上記方法は、その指定のメモリブロックにおいてスペアブロックが使用できないために上記メモリマネージャがブロック内のデータを圧縮しない

ようにスペアブロックのカウントを0にセットする段階を備えたことを特徴とする方法。

【請求項41】 コンピュータメモリマネージャにおいて指定の一回書き込み多数回読み取りのメモリデバイスがブロック消去可能でないことを指定する方法であって、上記コンピュータメモリマネージャはブロック消去可能な一回書き込み多数回読み取りのメモリデバイスをマネージするものであり、更に、上記メモリマネージャは、ブロック内のデータを圧縮するのに使用するためのスペアブロックのカウントを維持し、上記方法は、メモリマネージャによりブロックの消去を抑制するようにスペアブロックの上記カウントを0にセットする段階を備えたことを特徴とする方法。

【発明の詳細な説明】

【0001】

【産業上の利用分野】本発明は、一般に、ファイルをマネージするためのコンピュータシステムに係り、より詳細には、フラッシュ消去可能なプログラマブル・リードオンリメモリ（FEProm）に記憶されたファイルをマネージするための方法及びシステムに係る。

【0002】

【従来の技術】コンピュータシステムは、一般に、揮発性及び不揮発性の両方の記憶装置における情報の記憶をサポートする。揮発性記憶装置と不揮発性記憶装置の相違は、揮発性記憶装置から電源が切断されたときに情報が失われることである。これに対して、不揮発性記憶装置から電源が切断されたときには、情報が失われない。従って、不揮発性記憶装置に情報を記憶した場合には、たとえコンピュータシステムの電源が切られていても、ユーザはあるときに情報を入力しそしてその後に情報を検索することができる。又、ユーザは、不揮発性記憶装置をコンピュータから切断しそしてその記憶装置を別のコンピュータに接続して、その別のコンピュータが情報をアクセスするようにすることができる。

【0003】不揮発性記憶装置に記憶された情報は、一般に、ファイルに編成される。ファイルとは、関連情報を収集したものである。時間の経過と共に、コンピュータシステムは、記憶装置の容量にもよるが、記憶装置に何百、何千のファイルを記憶することができる。情報を記憶することに加えて、コンピュータシステムは、典型的に、ファイルの情報を読み取ったり、修正したり、削除したりすることができる。コンピュータシステムは、記憶、読み取り、修正及び削除を効率的に行えるように記憶装置にファイルを編成することが重要である。

【0004】一般にコンピュータのオペレーティングシステムの一部分であるファイルシステムは、記憶装置上のファイルを管理する助けをするために開発されたものである。1つのこのようなファイルシステムが、マイクロソフト・コーポレーションによりそのディスクオペレーティングシステム（MS－DOS）として開発されて

いる。このファイルシステムは、ファイルを記憶するためにハイアラーキ解決策を使用している。図1は、記憶装置のディレクトリ構造体を示している。これらのディレクトリはファイルの論理グループを含む。これらのディレクトリは、引出しの折り畳み器が引き出し内にペーパを編成していくのと同様にファイルを編成する。DOS、WORD、DAVID及びMARYと示されたブロックは、ディレクトリを表しており、そしてAUTOEXEC. BAT、COMMAND. COM、FORMAT. EXE、LETTER2. DOC、LETTER. DOCと示されたブロック及びLETTER1. DOCと示された2つのファイルは、ファイルを表している。このディレクトリ構造では、ユーザが関連ファイルをそれら自身のディレクトリに入れることによりファイルを編成することができる。この例において、WORDというディレクトリは、ワードプロセッシングプログラムWORDによって発生された全てのファイルを含む。このディレクトリWORDの中には2つのサブディレクトリDAVID及びMARYがあり、これらは、WORDファイルを、Davidにより開発されたもの及びMaryにより開発されたものへと更に編成する上で助けとなるものである。

【0005】従来のファイルシステムは、不揮発性記憶装置の多数回書き込み機能の利点を取り入れている。多数回書き込み機能は、記憶装置上のいかなる情報ビットでも実質上無限の回数で1から0へそして0から1へ変更することができる。この機能では、ファイルを記憶装置に書き込みそしてファイルの幾つかのビットを変更することによりファイルを選択的に修正することができる。

【0006】ディスクのような多数回書き込み能力を有する従来の記憶装置の欠点は、その速度が内部のコンピュータメモリの速度に比して遅いことである。これに対し、そのコンピュータメモリに勝るこれら記憶装置の利点は、不揮発性であること、コストが低いこと、容量が大きいことである。

【0007】フラッシュEProm（FEProm）として知られている記憶装置は、内部コンピュータメモリの速度と、コンピュータディスクの不揮発性とが結合されたものである。この装置はEProm型（消去可能なプログラマブル・リードオンリメモリ）のデバイスである。FEPromの内容は、典型的なEPromと同様にデバイスに紫外線を照射するのではなく、入力にある電圧を印加することによって消去することができる。消去は、デバイス内の各ビットを同じ値にセットする。他のEPromと同様に、FEPromは不揮発性メモリである。FEPromは、その速度がコンピュータの内部メモリに匹敵する。最初と、消去後とに、FEPromの各ビットは1にセットされる。FEPromの特徴は、他のEPromと同様に、ビット値1を0に変更で

きるが、ビット値0は1に変更できないというものである。従って、データは、1から0へのビットの変更を行うようにEPromに書き込むことができる。しかしながら、いったんビットが0に変更されると、1に変更し直すことはできず、即ち全EPromを全て1に消去しない限り1に戻すことができない。実際には、EPromの各ビットは一度しか書き込むことができないが、次々の消去の間に何回も読み取ることができる。更に、EPromの各ビットを消去して0にセットできるのは、ある限定された回数だけである。この限定された回数がEPromの実効寿命を定める。

【0008】EPromをアクセスする典型的な時間は、アクセスの形式と多数の他のファクタとによって異なる。読み取りアクセス時間は数百ナノ秒の範囲であり、バイトを読み取る回数については制限がない。書き込みアクセス時間は、典型的に数十マイクロ秒の範囲である。書き込みアクセス時間は、バイトを消去した回数と、デバイスの温度と、EPromのバイト密度とによって左右される。バイトを書き込む回数に理論的な制限はないが、消去の制限が実際上の書き込みの制限となる。EPromの消去時間は数秒の範囲である。消去時間は、EPromを消去した回数、デバイスの温度、及びEPromのバイト密度とによって左右される。

【0009】

【発明が解決しようとする課題】従来のファイルシステムは、記憶装置が多数回書き込み能力を有するものと仮定していたので、これらのファイルシステムは、実際上一回の書き込み能力しか持たないEPromには不適である。従って、EPromをベースとする記憶装置をサポートするファイルシステムをもつことが望ましい。このようなファイルシステムは、コンピュータシステムの速度と、コンピュータディスクの不揮発性とを有するものである。

【0010】コンピュータディスクのような従来の記憶装置は、バイトでアドレスできるのではなくブロックでアドレスすることができる。バイトは、コンピュータの内部メモリのアドレス能力の単位であり、即ちコンピュータは一度に1バイト（典型的に8ビット）で書き込み又は読み取りすることができるのであって、それ未満ではできない。コンピュータがディスクに書き込んだりディスクから読み取ったりするときには、ブロックと称するバイトのグループで行わねばならない。ブロックのサイズは変えられるが、典型的には2の累乗である（128、256、512等）。例えば、ディスク上の1バイトだけを変更しようとする場合に、そのブロックサイズ内の全バイト数の書き込みを行わねばならない。これには、ディスクからコンピュータメモリへブロック全体を読み取り、1バイトを変更し（内部メモリはバイトでアドレス可能である）そしてそのブロック全体をディスクに書き込むことが含まれる。

【0011】従来のファイルシステムは、ブロックに未使用の部分を残す状態でデータを記憶する。このファイルシステムは、一度に1つのファイルのみからのデータを所与のブロック内に記憶する。例えば、このファイルシステムは、1つのファイルからのデータをブロックの最初の5バイトには記憶せず、そして別のファイルからのデータを128バイトブロックの最後の7バイトには記憶しない。しかしながら、ファイルの長さがブロックサイズの倍数倍でない場合には、ブロック端のスペースが未使用となる。上記例において、ブロックの最後の7バイトは未使用となる。ディスクが4096といった大きなブロックサイズを使用するときには、4095バイトまでのデータが使用できないことになる。多数回書き込み能力を有していて数百万バイトを記憶できるようなディスクドライブではこのような未使用スペースが無視できる量であるが、多数回書き込み能力をもたず、しかも数百万バイトのデータを記憶する容量をもたない記憶装置では相当の量である。

【0012】EPromは、典型的な記憶装置に比して、ブロックアドレス式ではなくてバイトアドレス式である。従って、EPromのバイトアドレス能力をサポートするファイルシステムをもつことが望まれる。

【0013】又、EPromは、ブロックで消去可能なフォーマットで編成することができる。ブロックで消去可能なEPromは、個々に消去できる多数のブロック（典型的に16）を含んでいる。例えば、図6は、0から15まで番号が付けられた16個のブロックを有するブロック消去可能なEProm301の概略図である。ブロックの各々は、他のブロックの内容に影響を及ぼすことなく個々に消去することができる。ブロック番号14のブロック302は、他のブロックのデータに影響を及ぼすことなく消去することができる。ブロック消去可能なEPromをサポートするファイルシステムをもつことが所望される。

【0014】そこで、本発明の目的は、ファイル記憶装置、特に、ブロック消去可能で且つフラッシュ消去可能なプログラマブル・リードオンリメモリにデータを記憶する方法を提供することである。

【0015】本発明の別の目的は、ブロック消去可能なEPromにおいてメモリを割り当てたり割り当て解除したりするコンピュータメモリマネージャを提供することである。

【0016】本発明の別の目的は、ブロック消去可能なEPromにおいてブロックを消去した回数を追跡する方法を提供することである。

【0017】本発明の更に別の目的は、メモリの割り当てを容易にするデータ構造体を備えたブロック消去可能なEPromを提供することである。

【0018】本発明の更に別の目的は、データを記憶す

るためのブロックを割り当てる方法を提供することである。

【００１９】本発明の更に別の目的は、ブロック消去可能なＦＥＰｒｏｍにおいて割り当て解除されたスペースをリクレイミングする方法を提供することである。

【００２０】本発明の更に別の目的は、ブロック消去可能なＦＥＰｒｏｍのためのファイルシステムを提供することである。

【００２１】

【課題を解決するための手段】本発明の上記及び他の目的は、以下の説明から明らかとなるように、ブロック消去可能で且つフラッシュ消去可能なプログラマブル・リードオンリメモリにおいてメモリをマネージするための方法及びシステムを提供することにより達成される。このシステムは、好ましい実施例において、ブロックヘッダと、ブロック割り当てテーブルと、データ記憶エリアと、データを記憶すべきブロックを選択するためのブロック割り当てルーチンと、ブロック割り当てテーブルのエントリ及びデータ記憶エリアの一部分を選択するためのデータエリア割り当てルーチンと、データを記憶するための記憶ルーチンとを有したブロック消去可能なＦＥＰｒｏｍを具備している。又、このシステムは、好ましい実施例において、ブロック消去可能なＦＥＰｒｏｍのためのファイルシステムを実施するファイルマネージャを備えている。

【００２２】好ましい実施例において、本発明は、ブロック消去可能なＦＥＰｒｏｍのメモリをマネージする方法及びシステムを提供する。このシステムは、ＦＥＰｒｏｍマネージャ及びファイルシステムとして説明する。ＦＥＰｒｏｍマネージャは、ＦＥＰｒｏｍのメモリの割り当てと割り当て解除とを管理する。ファイルシステムは、ハイアラーキ式のディレクトリシステムであり、ＦＥＰｒｏｍマネージャを用いてメモリの割り当て及び割り当て解除を行う。或いは又、ＦＥＰｒｏｍマネージャ及びファイルシステムは、ある最適化を達成するように一体化することができる。しかしながら、個別のＦＥＰｒｏｍマネージャを使用することによりＦＥＰｒｏｍは異なるファイルシステムからのデータや非ファイルシステムデータも記憶することができる。

【００２３】

【実施例】本発明のＦＥＰｒｏｍマネージャは、ブロック消去可能なＦＥＰｒｏｍにおいて自由スペースの割り当て、割り当てられたスペースの割り当て解除、及び割り当て解除されたスペースのリクレイミングを行う。図２６に示す好ましい実施例では、ＦＥＰｒｏｍの各ブロックが、ブロック割り当て構造体２３０２と、データ領域２３０３と、自由スペース２３０４とを含んでいる。ブロック割り当て構造体２３０２は、ヘッダ及び割り当てアレイを含んでいる。ヘッダは、ブロックの状態情報を含む固定長さの構造体である。割り当てアレイは可変長さであり、割り当てアレイのエントリはデータ領域を記述する。テーブルＡは、ブロック割り当て構造体のデータ構造を示している。この構造体は、構造体変数の記述と共にＣプログラミング言語フォーマットで示されている。アレイのＡｌｌｏｃは、割り当てアレイであり、他の変数がヘッダを構成する。データ領域は、ＦＥＰｒｏｍに記憶されたデータを保持する可変長さの領域である。自由スペース２３０４は、ブロック割り当て構造体又はデータ領域に割り当てられないスペースである。ブロック割り当て構造体２３０２及びデータ領域２３０３はブロックの両端に割り当てられる。領域が追加されるときには、ブロック割り当て構造体２３０２及びデータ領域が矢印２３０５及び２３０６によって示されたように互いに向かって成長する。ブロック割り当て構造体のＡｌｌｏｃエントリは、ブロックの対応領域に対するオフセット２３１０－２３１５を含んでいる。好ましい実施例においては、ブロック割り当て構造体が、その領域に記憶されたデータに対して特定のデータを含んでいる。

【００２４】

テーブルＡ

データ構造体

```
struct BlockAllocation
 {
    struct
    {
      byte          Status;
      byte          Offset[3];
      word          Len;
    }               Alloc [ ];
    dword           BootRecordPtr
    dword           EraseCount;
    word            BlockSeq;
    word            BlockSeqChecksum;
    word            Status;
```

15 16

```
}
定義
Alloc                プロック内の領域を定める可変長さアレイ構造体
Status               領域の状態
    ビット#
    5-2        1111 未使用
               1011 中間状態
               0111 自由
               0011 割り当てられた
               0001 割り当て解除された
               0010 取って代わられた
               0000 ゼロ
    7-6          11 未使用エントリ
                 10 最終エントリ
                 00 非最終エントリ
Offset               この領域のブロックの開始に対するオフセット
Len                  領域の長さ( 単位バイト)
BootRecordPtr        FEPromがファイル記憶装置として使用されるとき
                     に記録をブートするための処理
EraseCount           ブロックを消去した回数
Blockseq             FEProm内のブロックの論理シーケンス
BlockSeqChecksum     ブロックシーケンス番号のチェック和
Status
    ビット#
    1-0          11 ブロックにないブート記録(FEProm がファイルシス
                    テムデータを含むとき)
                 10 ブロックのブート記録
                 00 取って代わられたブート記録
    15-10      110000 レディ
               0XXXXX QueuedForErasure
               111111 消去された
               111110 UpdateInProcess
               111100 スペアブロック
               111000 ReclaimationInProcess
               000000 リアタイアされた
```

【００２５】ＦＥＰｒｏｍマネージャは、ＡｌｌｏｃエントリをＩ追加し、自由スペース内の第１位置を指すように可変オフセットをセットし、変数Ｌｅｎを領域の長さにセットし、そして割り当てられたことを指示するように変数Ｓｔａｔｕｓをセットすることによりブロック内のデータ領域を割り当てる。ＦＥＰｒｏｍマネージャは、対応するＡｌｌｏｃエントリにおける変数Ｓｔａｔｕｓを割り当て解除された状態にセットすることにより領域の割り当て解除を行う。割り当て解除されたスペースは一般に再割り当てには使用できない。というのは、非ＦＮＵＬＬ値にセットされているからである。割り当て解除されたスペースは、再割り当てされる前にＦＮＵＬＬにセットされる。割り当て解除されたスペースをＦＮＵＬＬにセットしそして割り当てに使用できるようにするプロセスは、ブロックリクレイミングである。割り当て解除されたスペースは、割り当てられた領域を第２のブロックにコピーしそして第２のブロックのＡｌｌｏｃエントリを新たなオフセットを指すようにセットすることによりリクレイミングされる。ブロックのリクレイミングプロセスは、第２ブロックのＡｌｌｏｃエントリがブロックヘッダに対しコピー元ブロックにあったのと同じ位置になるように確保する。ＦＥＰｒｏｍマネージャは、論理ブロックシーケンス番号であるヘッダの変数ＢｌｏｃｋＳｅｑを使用して、データの特定の論理ブロックを識別する。リクレイミング中に、ブロックがコピーされるときには、物理的なブロック番号が変化するが、論理的なブロックシーケンス番号は変化しない。

【００２６】ＦＥＰｒｏｍマネージャは、領域のスタートアドレスではなくて割り当てられた領域に対するハンドルを与える。このハンドルは、２つの部分、即ち

（１）物理的なブロックを間接的に参照する論理ブロックシーケンス番号と、（２）物理的なブロック内の領域を間接的に参照するＡｌｌｏｃエントリに対するインデックスとを含んでいる。ハンドルは、論理的なブロックシーケンス番号に対応する物理的なブロックを決定しそしてハンドルのインデックス部分によって指示されたＡｌｌｏｃエントリの変数Ｏｆｆｓｅｔをアクセスしてその変数Ｏｆｆｓｅｔを物理的なブロックのスタートアドレスに加えることにより参照解除される。この参照解除により領域のアドレスが形成される。ハンドルを使用すると、ＦＥＰｒｏｍマネージャは、存在するかもしれないデータ領域へのリンクを調整せずにブロックをリクレイミングすることができる。更新しなければならないものは、メモリ内にあって論理ブロックシーケンス番号を物理的なブロックに対してマップする変換キャッシュ（以下で説明する）と、Ａｌｌｏｃアレイのオフセットだけである。ハンドルが参照解除されるときには、新たなオフセットを用いて正しいアドレスが形成される。図２９は、９の論理ブロックシーケンス番号２５０２と、３のＡｌｌｏｃインデックス２５０３とを有するハンドル２５０１の参照解除を示している。ブロック変換キャッシュ２５０４は、論理的なブロック番号を物理的なブロック番号に対してマップする。ＦＥＰｒｏｍマネージャはキャッシュを維持する。ある物理的なブロックがブロックのリクレイミング中に別の物理的なブロックへ移動されるときには、ＦＥＰｒｏｍマネージャは、論理ブロックシーケンス番号を新たな物理的ブロックにマップするように変換キャッシュを調整する。図２９の例においては、論理ブロックシーケンス番号９が物理的ブロック１４　２５０６へとマップされる。物理的なブロック番号１４に対するＡｌｌｏｃアレイは、ハンドル２５０１のＡｌｌｏｃインデックス２５０３によって指示される。Ａｌｌｏｃ［３］エントリの変数Ｏｆｆｓｅｔは、物理的なブロック番号１４　２５０５に領域３　２５０６のオフセットを含んでいる。領域３　２５０６のアドレスは、物理的なブロック２５０５のアドレスにオフセットを追加することによって決定される。

【００２７】図３３は、ブロック内に新たな領域を割り当てる領域割り当てルーチンの流れ線図である。このルーチンに対する入力パラメータは、その領域に記憶すべきデータと、データの長さである。このルーチンは、割り当てられた領域にハンドルを返送する。或いは又、このルーチンはデータを書き込まず、単にスペースを割り当てる。このルーチンは、ブロックが領域に対して充分な自由スペースと、追加のＡｌｌｏｃエントリとを有すると仮定する。図３５は、Ａｌｌｏｃエントリのステータスに対する状態図である。Ａｌｌｏｃエントリは、次のステータスのうちの１つである。即ち、未使用、割り当てられた状態、割り当て解除された状態、取って代わられた状態、自由、又はゼロである。又、エントリは、

割り当てが処理中である間は移行状態にある。エントリが未使用である場合には、Ａｌｌｏｃアレイのエントリを通過しており、即ちアレイの一部分ではない。エントリが割り当てられた場合には、それに対応する領域が現在割り当てられる。エントリが割り当て解除される場合には、それに対応する領域が不要であるが、その領域はまだリクレイミングされていない。エントリが取って代われる場合には、別のＡｌｌｏｃエントリ（単数又は複数）がその同じデータ領域に対応する。リクレイミング中、取って代われたエントリにあるデータは無視される。というのは、別のエントリ（単数又は複数）がデータ領域を指すからである。エントリが自由の場合には、それに対応する領域がリクレイミングされており、新たな領域がブロックに追加されたときにＡｌｌｏｃエントリを再使用することが出来る。エントリがゼロの場合には、エントリへの書き込み中に問題が生じており、消去されるまで使用されない。エントリが割り当てにおいてプロセス移行状態にある場合には、エントリにおけるデータの若干は有効であるが必ずしも全部ではない。

【００２８】図３５を参照すれば、全てのエントリは最初は未使用状態３１０１にあり、ブロックが消去されたときに未使用状態３１０１へ移行する。エントリは、未使用状態３１０１から割り当て進行状態３１０４を経て割り当て状態３１０３へ移行する。自由状態３１０２にあるエントリは割り当て状態３１０３へ移行する。未使用状態３１０１にあるエントリは、そのエントリが割り当て解除されるか又は取って代われて、リクレイミングされるべきブロック内の最後に割り当てられたエントリの後に探索されないときに、ブロックリクレイミングによって自由状態３１０２へ移行する。割り当てられた状態３１０３にあるエントリは、対応する領域が割り当て解除されたときに割り当て解除状態３１０５に移行する。割り当てられた状態３１０５にあるエントリは、同じ領域に対応する別の１つ又は複数のエントリが割り当てられたときに、取って代われた状態３１０６へ移行する。最後に、いかなる状態にあるエントリも、そのエントリに対する書き込みエラーの際にはゼロ状態３１０７へ移行する。

【００２９】図３３を参照すれば、ブロック２９０１において、システムはＡｌｌｏｃエントリが自由の状態にあるかどうかを判断する。このようなエントリが見つかった場合には、システムはそのエントリを再使用し、ブロック２９０２へ続くが、さもなくば、ブロック２９０３へ続く。ブロック２９０２において、システムは自由のＡｌｌｏｃエントリを選択し、ブロック２９０５へ続く。ブロック２９０３において、システムは新たなＡｌｌｏｃエントリを選択し、ブロック２９０４へ続く。新たなＡｌｌｏｃエントリは、最終とマークされたエントリの直後のエントリである。ブロック２９０４におい

て、システムは、選択されたＡｌｌｏｃエントリに対して変数Ｓｔａｔｕｓをセットし、割り当てが進行中であることを指示する。割り当て進行中状態は、データが一貫した状態にないかもしれないことを指示する移行状態である。ブロック２９０５において、システムは変数Ｏｆｆｓｅｔ及びＬｅｎをセットし、自由スペースの第１位置から始まってデータ領域にデータを書き込む。ブロック２９０６において、ＡｌｌｏｃエントリがＮＥＷであった場合には、システムはブロック２９０７へ続くが、さもなくば、システムはブロック２９０８へ続く。ブロック２９０７において、システムは手前の最終Ａｌｌｏｃエントリのステータスをリセットし、もはやＡｌｌｏｃ構造体の最終エントリでないことを指示する。ブロック２９０８において、システムは、選択されたＡｌｌｏｃエントリに対する変数Ｓｔａｔｕｓを割り当てられた状態にセットし、データが今や一貫した状態にあることを指示する。次いで、システムは割り当てを終了する。

【００３０】上記したように、ＦＥＰｒｏｍのブロックの性能は、消去カウントが増加するにつれて低下する。各ブロックの消去カウントをほぼ等しく（均等化したと称する）維持することが好ましい。通常の動作においては、消去カウントが均等化されない。例えば、実行可能なファイルがブロックに書き込まれた場合には、そのブロックを消去する必要は決して生じない。従って、そのブロックの消去カウントは、他のブロックの消去カウントが増加しても一定のままである。好ましい実施例ではブロックの消去カウントを均等化するために多数の解決策を使用する。まず、ブートアップ中又はＦＥＰｒｏｍがロードされるときに（初期化）、ＦＥＰｒｏｍマネージャはブロックを走査して、ＦＥＰｒｏｍをマネージするデータを得る。好ましい実施例では、このデータが各ブロックの消去カウントを含んでいる。消去カウントの均等化を助けるために、ＦＥＰｒｏｍマネージャは、消去カウントの大きいブロックのデータと、消去カウントの低いブロックのデータをスワッピングする。このスワッピングは時間がかかるので、初期化中に生じるスワッピングの回数を最小にすることが望ましい。更に、スワッピングは、消去カウントの差がスレッシュホールド値又は比を越えたときだけ実行すればよい。第２に、ＦＥＰｒｏｍマネージャがブロックに対してリクレイミングを実行するときには、消去カウントの小さい使用可能なブロックを選択するのが好ましい。第３に、ＦＥＰｒｏｍマネージャが領域を割り当てるときには、消去カウントの低いブロックの領域を割り当てる。第４に、ＦＥＰｒｏｍマネージャはブロックの消去回数を追跡する。消去の回数がスレッシュホールド値を越えた場合には、ＦＥＰｒｏｍマネージャは、２つのブロックに対する消去カウントの差がスレッシュホールド値又は比を越えたかどうかを決定する。そのスレッシュホールド値を越えた場合には、マネージャはそれらブロックのデータをスワッピングする。

【００３１】ＦＥＰｒｏｍマネージャは、各物理的なブロックのヘッダに消去カウントを維持するのが好ましい。ブロックが消去されたときには、ＦＥＰｒｏｍマネージャは、増加された消去カウントをブロックヘッダに書き込んで戻す。ブロックがコピーされたときには、消去カウントが転送されない。各ブロックはそれ自身の消去カウントを保持する。或いは又、消去カウントは単一のブロックに記憶することができる。しかしながら、この別の方法は、好ましい方法以上に幾つかの欠点がある。第１に、単一のブロックに欠陥が生じると、全ての消去カウントが失われる。第２に、ブロックが消去されると、消去カウントブロックを更新しなければならない。最終的に、消去カウントブロックは消去され、再書き込みされねばならない。

【００３２】図３２は、ブロック消去可能なＦＥＰｒｏｍのどのブロックを使用して領域を割り当てるかを選択するブロック割り当てルーチンの流れ線図である。ＦＥＰｒｏｍマネージャは、多数のファクタに基づいてどのブロックを割り当てるかを決定する。まず、ＦＥＰｒｏｍマネージャは、消去カウントが最も低い充分な自由スペースを有するブロックを割り当てる。消去カウントの低いブロックを割り当てることは、ブロックの均等化を確保する上で助けとなる。第２に、ＦＥＰｒｏｍマネージャは、充分な自由スペースをもつブロックがない場合には多数のブロックを割り当てる。データは、異なるブロックに各々記憶された多数の領域に分割される。第３に、ＦＥＰｒｏｍマネージャは、あまりに多数の部分が生じるときには割り当てを許さない。このルーチンに対する入力パラメータは、割り当てられるべき領域の長さと、領域を多数のブロックに記憶できるかどうかである。ブロック２８０１において、システムは、データを記憶するに充分な自由スペースを有する全てのブロックを選択する。好ましい実施例において、システムは、自由スペースの開始位置とＡｌｌｏｃエントリの数とに基づいて自由スペースの長さを決定する。このデータは、初期化中及び必要に応じて更新される間にＦＥＰｒｏｍマネージャバッファに記憶されるのが好ましい。又、システムは、もし必要ならば、新たなＡｌｌｏｃエントリを追加するに充分なスペースが存在するよう確保する。１つの実施例において、システムは、ブロックが充分な自由スペースを有しているかどうかを決定する前に、リクレイミング基準に合致するブロックに対してブロックリクレイミングを実行する。別の実施例においては、充分な自由スペースがないと決定されるまでリクレイミングは行われない。ブロック２８０２においては、少なくとも１つのブロックが選択された場合に、領域を保持するに充分な自由スペースが単一ブロックにあり、システムはブロック２８０３に続き、さもなくば、システムはブロック２８０４に続く。ブロック２８０３において、

システムは、選択されたブロックのどれが最も低い消去カウントを有するかを決定し、そのブロックを割り当てると共に、システムはブロックの割り当てを終了する。ブロック２８０４においては、システムは、領域データを保持するに充分な自由スペースを有する１組のブロックを選択する。ブロック２８０５においては、全自由スペースと割り当て解除されたスペースがデータを保持するに充分でないか或いはあまりに多数の部分がある場合に、システムはブロック２８０７へ続き、さもなくば、システムはブロック２８０６へ続く。領域データを単一ブロックに記憶しなければならない場合には、２つのブロックを選択すると、部分が多数になり過ぎる。又、データを多数のブロックに記憶すべき場合には、リクレイミングが適当である。ブロック２８０６において、システムは選択されたブロックを割り当て、ブロックの割り当てが終了する。ブロック２８０７において、全てのブロックがリクレイミングされた場合には、ＦＥＰｒｏｍにはデータを記憶するに充分な余裕がなく、ブロックの割り当てが終了するか、さもなくば、システムはブロック２８０８へ続く。ブロック２８０８において、システムはブロックをリクレイミングし、ブロック２８０１へ進んで、新たな充分な自由スペースがあるかどうかを決定する。

【００３３】図３４は、ブロック内の割り当て解除された領域をリクレイミングするブロックリクレイミングルーチンの流れ線図である。入力パラメータは、リクレイミングされるべきブロックの数と、スペアブロックの物理的な番号とである。ブロックは、多数の種々の時間にリクレイミングされる。第１に、割り当て要求を満たす充分な自由スペースがないときには、ブロックがリクレイミングされる（上記したように）。第２に、ＦＥＰｒｏｍマネージャは、ＦＥＰｒｏｍへの書き込みアクセスの回数を追跡することができる。書き込み回数がスレッシュホールド数を越えた場合には、マネージャがいずれかのブロックをリクレイミングすべきかどうかを決定する。割り当て解除されたスペースとブロックサイズとの比がスレッシュホールド値を越えたときにはブロックをリクレイミングしなければならない。当業者に明らかなように、例えば、ＦＥＰｒｏｍが最初にロードされるときのような別の時間にブロックリクレイミングを行うこともできる。ＦＥＰｒｏｍマネージャは、割り当てられた領域をスペアブロック、即ち消去されたブロックへコピーすることによりブロックをリクレイミングする。割り当てられた領域のみをコピーすることにより、割り当て解除された領域がリクレイミングされる。或いは又、ＦＥＰｒｏｍマネージャは、割り当てられた領域を非ＦＥＰｒｏｍメモリへコピーし、次いで、ブロックを消去し、そして領域をブロックへコピーして戻す。しかしながら、この方法は、割り当てられた領域を記憶するに充分な非ＦＥＰｒｏｍメモリを必要とし、消去の後であって

て且つブロックが再書き込みされる前に停電が生じたとすれば、データを失うおそれがある。好ましい方法においては、ＦＥＰｒｏｍマネージャは、リクレイミングされるべきブロック内の割り当てられた領域をスペアブロックへコピーすると共に、そのスペアブロック内の新たな領域位置を表す変数Ｏｆｆｓｅｔを調整するブロック割り当て構造体をコピーする。図２７は、領域１及び５をリクレイミングした後の図２６のブロックレイアウトの例を示している。割り当てられた領域は隣接するようにコピーされている。対応するＡｌｌｏｃエントリは、新たな領域位置を指すように更新される。たとえ領域１がリクレイミングされていてもＡｌｌｏｃ〔１〕エントリは依然として必要とされる。ハンドルを使用するために、全てのＡｌｌｏｃエントリはブロック割り当て構造体においてそれらの同じ位置を維持しなければならない。しかしながら、Ａｌｌｏｃ〔１〕エントリに対する変数Ｓｔａｔｕｓは自由状態にセットされ、これを用いて、リクレイミングされたブロックに追加される次の位置を指すことができる。Ａｌｌｏｃ〔５〕エントリの後にはＡｌｌｏｃ入力はないので、ハンドルに対するプレースホルダーとして必要とされず、除去されている。Ａｌｌｏｃ〔４〕エントリの位置状態は、それがＡｌｌｏｃアレイにおける最終エントリであることを指示する。

【００３４】図３６はブロックの状態を示す状態図である。ブロックの状態は変数Ｓｔａｔｕｓのヘッダに記憶される。ブロックの状態は、消去３２０１、更新進行中３２０２、スペア３２０３、リクレイミング進行中３２０４、レディ３２０５、消去のための待ち行列３２０６、及びリタイア３２０７である。新たに消去されたブロックは消去状態３２０１である。消去されたブロックは、通常、更新進行状態３２０２へ移行され、次いで、スペア状態３２０３へ移行される。更新進行状態３２０２は、ヘッダ内のあるデータ、例えば消去カウントが更新されていることを指示する。この更新が完了すると、ブロックはスペアの状態３２０３へ移行する。更新が失敗すると、ブロックは消去のための待ち行列状態３２０６へ移行する。スペア状態３２０３のブロックは、そのブロックがリクレイミングされているブロックからデータを受け取るべきときに、リクレイミング進行状態３２０４へ移行する。リクレイミング進行状態３２０４は、ブロック割り当て構造体が一貫した状態にないことを指示する移行状態である。データが一貫したものになると、ブロックはレディ状態３２０５へ移行する。しかしながら、リクレイミング進行状態３２０５の間にエラーが生じた場合には、そのブロックに対してリクレイミングが行われた後に、ブロックが消去待ち行列状態３２０６へ移行する。消去待ち行列状態３２０６にあるブロックは、消去されたときに消去状態３２０１へ移行する。消去が失敗すると、ブロックはリタイア状態３２０７へ移行する。それからブロックはリタイア状態に留まる。

ＦＥＰｒｏｍが最初に初期化されるときには、ブロックがレディ状態又はスペア状態にセットされる。レディ状態にあるブロックはデータを含むことができ、スペア状態にあるブロックはデータを含まない。

［００３５］図３４を参照すれば、ブロック３００１において、システムは、リクレイミングされるべきスペアブロックに対する変数Ｓｔａｔｕｓを、リクレイミング進行中を指示するようにセットする。ブロック３００２において、システムは、リクレイミングされるべきブロックのヘッダから変数Ｓｅｑ、ＳｅｑＣｈｅｃｋＳｕｍ、及びＢｏｏｔＲｅｃｏｒｄＰｔｒをスペアブロックへコピーする。ブロック３００３において、システムは、リクレイミングされるべきブロックに対して割り当て状態にある最終Ａｌｌｏｃエントリの位置を決定する。リクレイミングプロセス中に、Ａｌｌｏｃエントリは、最後に割り当てられたエントリの後に無視される。従って、リクレイミングされたブロックは、これらエントリの状態を未使用にセットする。ブロック３００４ないし３０１０において、システムは、最後に割り当てられたエントリまでの各Ａｌｌｏｃエントリをスペアブロックにコピーする。ブロック３００４において、システムは、Ａｌｌｏｃエントリを指示するインデックスｊを初期化する。ブロック３００５において、このインデックスｊがブロック３００３で決定された最後に割り当てられたエントリのインデックスより大きい場合には、全てのＡｌｌｏｃエントリとそれに対応する領域がコピーされており、システムはブロック３０１１へ続き、さもなくば、システムはブロック３００６へ続く。ブロック３００６において、エントリの状態が割り当てられた状態である場合には、システムはブロック３００７へ続き、さもなくばブロック３００９へ続く。ブロック３００７において、システムは、Ａｌｌｏｃ〔ｊ〕エントリに対応する領域データをスペアブロックへコピーする。ブロック３００８において、システムは、スペアブロックのＡｌｌｏｃ〔ｊ〕入力における変数Ｏｆｆｓｅｔを更新し、コピーされた領域の位置を指示するようにすると共に、変数Ｓｔａｔｕｓ（位置の状態を適当にセットする）及びＬｅｎをコピーし、ブロック３０１０に続く。ブロック３００９では、システムは、スペアブロックのＡｌｌｏｃ〔ｊ〕エントリの状態を更新し、そのエントリが自由状態であることを指示するようにする。ブロック３０１０では、システムはインデックスｊを増加して、次のＡｌｌｏｃエントリを指示し、ブロック３００５へとループする。ブロック３０１１においては、システムはスペアブロックの状態をレディにセットする。ブロック３０１２では、システムは、リクレイミングされるべきブロックの状態を消去のための待ち行列にセットする。ブロック３０１１についての処理が完了した後であって且つブロック３０１２についての処理が完了する前に、スペアブロックと、リクレイミングされるべきブロックの両方は有効データを有している。ブロック３０１２についての処理が完了する前に処理が中断した場合には、ＦＥＰｒｏｍは、同じ論理シーケンス番号を有する２つのブロックを含む。システムは、好ましくは、ＦＥＰｒｏｍの初期化中にこの状態をチェックする。このときに、システムはブロック３０１２の処理を完了することができる。ブロック３０１３では、システムは、スペアブロック（以下で述べる）の状態を表すように変数ＰｈｙｓｉｃａｌＢｌｏｃｋＮｕｍ、ＦｉｒｓｔＦｒｅｅＢｙｔｅＯｆｆｓｅｔ、ＬｅｎＤｅａｌｌｏｃａｔｅＳｐａｃｅ及びＡｌｌｏｃＳｔｒｕｃｔＣｎｔ ｉｎ ＢｌｏｃｋＤａｔａ〔Ｓｅｑ〕を更新する。ブロック３０１４では、システムは、スペアブロックのリストを調整するようにＤｒｉｖｅＲｅｃを更新し、リクレイミングが終了となる（以下で述べる）。

【００３６】

表 Ｂ

```
データ構造
struct      DriveRec
 {   word   BlockCnt
     word   SpareBlockCnt
     dword  BlockSize
     word   RootDirPtr
     word   SpareBlockPtr 〔 〕
 }
 struct     ConfigRec
  {
     word   WriteAccessCntThreshold
     word   EraseCntThreshold
     word   BlockReclamationThreshold
     word   BlockEraseLevelingThreshold
 }
 struct     BlockRec
  {
     byte   Flags
     word   PhysicalBlockNum
     dword  FirstFreeByteOffset
     dword  LenDeallocatedSpace
     word   AllocStructCnt
     dword  BlockEraseCnt
     word   FirstUseableAllocEntry
     word   FreeAllocEntryCnt
 }   BlockData 〔 〕
定義
BlockCnt                    FEProm内の物理ブロックの数
BlockSize                   ブロック内のバイトの数
RootDirPtr                  FEPromをファイル記憶装置として使用した時のルートディレクトリを含むデー
```

タ領域へのハンドル

SpareBlockPtr [ ] 予備ブロックの物理ブロックの数を含む可変長アレイ

SpareBlockCnt 予備ブロックの数

WriteAccessCntThreshold ブロックを再利用すべきか否かをシステムに決定させる FEProm への書込みの数

EraseCntThreshold ブロックレベル化を行うべきか否かをシステムに決定させる FEProm への消去の数

BlockReclamationThreshold ブロック再利用をトリガする割当て解除スペースとブロックサイズとの比

BlockEraseLevelingThreshold ブロック間のレベル化プロセスをトリガする最小及び最大消去数間の差

PhysicalBlockNum 論理的ブロックを含む物理ブロックの数

FirstFreeByteOffset 空きスペースの最初のバイトの物理ブロック内のオフセット

LenDeallocatedSpace 物理ブロック内の割当て解除済領域の合計の長さ

FirstUseableAllocEntry ブロック内の最初の使用可能な Allocエントリ索引

FreeAllocEntryCnt Alloc エントリの数

BlockEraseCnt 物理ブロックが消去された回数

FEPromが最初にロードされる時、FEProm管理者は FEProm を走査して表Bに示す内部データを初期化する。構造 DriveRec は装置に関するデータを含み、構造ConfigRec は構成パラメタに関するデータを含み、アレイ BlockDataは FEProm内の各物理ブロック毎のデータを有するエントリを含む。アレイ BlockDataはブロック変換キャッシュである。初期化中、 FEProm 管理者はアレイ BlockData内の各変数及び構造 DriveRec 内の予備ブロックに関する変数を初期化する。構造DriveRec内の他の変数はシステム定義された変数である。好ましい実施例では、FEProm管理者は、これらのデータ構造内の領域データの型に特定の情報を記憶している。例えばもし FEProm がファイル記憶装置として使用されていれば、データ構造はルートディレクトリへのハンドルを含むことができる。図３１は、好ましい実施例における初期化プロセスの流れ図である。この手順は、 FEProm 上の各ブロック毎にブロック割当て構造を走査することによって DriveRec 及び BlockData構造を初期化する。システムはブロック２７０１乃至２７０９をループして各ブロック毎のデータを読む。ブロック２７０１ではシステムは、現在アクセスされている物理ブロックを指示する索引ｉを初期化する。ブロック２７０２において、もし索引ｉが FEProm 内のブロックの数より大きければ、全てのブロックは走査されたのでありシステムはブロック２７１０へ進み、そうでない場合にはシステムはブロック２７０３へ進む。ブロック２７０３では、システムは索引ｉによって指示されたブロックの見出しを読む。ブロック２７０４では、システムは DriveRec 及び BlockData〔ｉ〕データを更新する。もしそのブロックが予備ブロックであれば、システムは SpareBlockCntをインクリメントさせ、そのブロックを SpareBlockPtrアレイに追加する。好ましい実施例では、システムはこれらの領域内に配憶されているデータに特定の情報をも走査する。例えばもし FEProm をファイルシステムとして使用していて、そのブロックがブートレコードを含んでいれば、システムは BootRecPtr をセットし、ブートレコードを読み、そして RootDirPtr をセットする。

【００３７】システムはブロック２７０５乃至２７０８をループして各 Allocエントリ内のデータを処理する。ブロック２７０５では、 Allocエントリを指示する指標ｊを初期化する。ブロック２７０６において、もしシステムが最後のAlloc エントリを処理済であればシステムはブロック２７０９へ進み、そうでなければシステムはブロック２７０７へ進む。ブロック２７０７では、システムはｊによって指示された Allocエントリに基づいて BlockData〔 BlockSeq 〕データを更新する。システムは変数 FirstFreeByteOffset、LenDeallocatedSpace 、及び AllocaStructCntを更新する。システムは、変換キャッシュを初期化する索引ｉに変数 PhysicalBlockNum をセットする。ブロック２７０８では、システムは索引ｊをインクリメントさせて次の Allocエントリを指示させ、ブロック２７０６へループする。ブロック２７１０では、システムはブロック使用をレベル化する。システムは BlockDataアレイを走査して最大 BlockEraseCntを有するブロック及び最小 BlockEraseCntを有するブロックを決定する。次いでシステムはブロック間でデータをスワップ（交換）する。システムは先ず最大ブロックを予備ブロックへ複写する。次にシステムは最大ブロックを消去する。システムは最小ブロックからのデータを消去されたブロックへ複写し、そして好ましくは複写しながらブロック再利用を遂行する。システムは最小ブロックを消去し、予備ブロックからのデータを最小ブロックへ複写し、そして好ましくは複写中にブロック再利用を遂行する。

【００３８】本発明の FEProm 管理者はブロック化されていない、または消去することができない媒体を支援することができる。ブロック再利用及びブロック消去カウントレベル化プロセスはブロック消去可能性に頼っている。従ってもし媒体がブロック消去可能性を支援しなければ、これらのプロセスを不能にすべきである。好ましい実施例では、予備ブロックカウントを０にセットすることによってこれらのプロセスを実効的に不能にすることができる。 FEProm 管理者はこれらのプロセスを作動させるために少なくとも１つの予備ブロックに頼っている。もし媒体がブロック化されていなければ、任意の

ブロックサイズを論理的ブロックとして選択することができる。好ましい実施例ではブロックのサイズは、割当てアレイエントリ内のオフセットが全ブロックをアドレスできない程大きくすべきではなく、またブロック見出し及び割当てアレイがブロックサイズのかなりなパーセントを占めたり、または変換キャッシュが大き過ぎる程は小さくすべきでない。

【００３９】FEProm 管理者は、 FEProm 書込み及び消去エラーからの動的な回復を可能にする。書込みエラーは、メモリ位置を指定された値にセットできない場合に生成される。これらのエラーは、ハードウエアの障害によって、または既に０に変化しているあるビット内に１を要求するというように、あるメモリ位置にある値を書込もうと試みることによってもたらされる。

【００４０】書込みエラーはデータ領域、ブロック見出し、及び割当てアレイエントリへ書込む時に発生し得る。好ましい実施例では、データ領域へ書込み中に書込みエラーが発生すると、 FEProm 管理者はその1ブロックを割当て解除された状態にセットする。次いで FEProm 管理者は、図３３に示す領域割当てプロセスを再始動させることによって、データを異なるデータ領域へ書込むことを試みる。

【００４１】割当てアレイエントリへ書込み中に書込みエラーが発生すると、 FEProm 管理者はその割当てアレイエントリを空白（ヌル）状態にセットする。あるエントリを置換された（ superseded ）状態、割当て解除された状態、解放された（フリー）状態、または割当て進行中状態にセットしている時に、もし書込みエラーが発生すると、そのエントリを空白状態にセットすることによって FEProm はこれを（ consistent ）状態に保たれる。しかしながら、もしあるエントリを割当て済状態にセットしている時にエラーが発生すると、データ領域は割当て済状態の対応割当てアレイエントリを有していないのであるから、 FEProm は非一貫状態になる。 FEProm 管理者はそのデータ領域に対して別のエントリを割当てる。あるエントリを空白状態にセットしている時にもエラーは発生し得る。空白状態は、０の値のステータスとして定義されているから、あるエントリを空白状態にセットしている時に発生するエラーは必然的にハードウエアエラーである。もしエラーが発生すれば、対応する領域を再利用しなければならないことを指定する再利用を必要とするかも知れない。例えば、もしあるエントリを割当て解除状態にセット中に、及び再びそのエントリを空白状態にセット中にエラーが発生すれば、そのエントリは割当て済状態になる。もしこの状態のままであれば、このエントリ及び対応データ領域は正常の再利用の下では決して再利用されなくなる。

【００４２】ブロック見出しに書込み中にエラーが発生すると、 FEProm 管理者はそのブロックを消去待ち行列状態にセットする。もしあるブロックを消去待ち行列状態にセット中にエラーが発生すれば、 FEProm 管理者はそのブロックを引退（ retired）状態にセットするので、そのエラーを回復することはできなくなる。あるブロックを消去中に書込みエラーが発生すると、 FEProm 管理者はそのブロックを引退状態にセットする。もし引退させたブロックが予備ブロックであったならば、 FEProm 管理者は少なくなった予備ブロックで動作する。代替としてFEProm管理者は割当て済の割当てアレイエントリを用いずにブロックを探知することを試みる。次いで FEProm 管理者は探知したブロックを消去し、それを予備状態にセットする。予備ブロックが使用できない場合には FEProm 管理者は FEProm を、あたかもそれが前述した消去不能であるかのように取扱う。

【００４３】本発明は、 FEProm が非一貫状態にある場合の動的エラー回復をも提供する。図３３を参照する。例えば、もしブロック２９０５においてオフセットが書込まれた後に、しかしブロック２９０８において状態が解放状態から割当て済状態へ更新される前に FEProm が除去されれば、 FEProm は非一貫状態になる。FEPromに次にロードする時に、 FEProm 管理者は割当てエントリが解放されていることを見て、それを再使用しようとする。しかしながら、オフセットへ書込もうとする試みは失敗する（同一データがオフセットへ書込まれている場合を除く）。上述したように、 FEProm 管理者はエントリを空白状態にセットすることによって回復し、領域割当てプロセスを再始動させて異なるエントリを選択する。もしFEPromが取外される前にデータの一部分がデータ領域へ書こまれれば、 FEProm 管理者がデータをその領域へ書込もうとする時にエラーが検出される。このエラーは上述したようにして処理される。

ファイルシステム

本発明は、 FEProm 装置に対してディレクトリをベースとする階層ファイルシステムを提供する。階層ファイルシステムは論理的グループ内にファイルを配憶させる。好ましい実施例は、ディレクトリ階層及び内部ファイル配憶の両者を実現するためにリンクされたリストデータ構造を使用する。

【００４４】図１に典型的な階層ディレクトリ構造を示す。ワシントン州レッドモンドのマイクロソフト社から入手できる MS-DOS オペレーティングシステムが、階層ディレクトリ構造を有するファイルシステムを実現する。図１に示すように、ディレクトリ「ルート」１００は、２つのサブディレクトリ（「 DOS」１０２及び「ワード」１０３）と、２つのファイル（「 AUTOEXEC.BAT 」１０４及び「 COMMAND.COM」１０５）とを含む。ディレクトリ「 DOS」１０２は１つのファイル（「 FORMAT.EXE 」１０６）を含む。ディレクトリ「ワード」１０３は次に低いレベルに２つのサブディレクトリ（「デービッド(DAVID) 」１０７及び「メリー (MARY) 」１０８）を含む。ディレクトリ「デービッド」１０７は１つ

のファイル「LETTER1.DOC」１０９を含む。ディレクトリ「メリー」１０８は３つのファイル「LETTER1.DOC」１１０、「LETTER2.DOC」１１１及び「LETTER3.DOC」１１２を含む。

【００４５】図２は、好ましい実施例において図１のディレクトリ構造を実現する考え得るリンクされたリストを示す。ディレクトリ「ルート」レコード１００（本明細書においては「レコード」及び「エントリ」という語を互換可能なように使用する）はポインタ１２０を有し、このポインタ１２０は次に低いレベルのサブディレクトリのリンクされたリスト１４０及びファイルレコードを指し示す。リンクされたリスト１４０は、ポインタ１２１、１２２、１２３によってリンクされたサブディレクトリレコード DOS１０２及び「ワード」１０３と、ファイルレコード「AUTOEXEC.BAT」１０４及び「COMMAND.COM」１０５とからなる。サブディレクトリレコード「DOS」１０２は次に低いレベルのファイルレコード１０６を指し示すポインタ１２４を有し、サブディレクトリレコード「ワード」１０３は次に低いレベルのファイルレコードのリンクされたリスト１４１を指し示すポインタ１２５を有している。リンクされたリスト１４１は、ポインタ１２６によってリンクされているディレクトリレコード「デービッド」１０７及び「メリー」１０８からなる。サブディレクトリレコード「デービッド」１０７は次に低いレベルのファイルを指し示すポインタ１２７を有し、サブディレクトリレコード「メリー」１０８は次に低いレベルのファイルレコードのリンクされたリスト１４２を指し示すポインタ１２８を有している。リンクされたリスト１４２は、ポインタ１２９及び１３０によってリンクされているファイルレコード「LETTER1.DOC」１１０、「LETTER2.DOC」１１１及び「LETTER3.DOC」１１２からなる。右上のテンプレート１０は図面を通して使用されるレコードレイアウトを示す。好ましい実施例では図２に示すレコードは、以下に説明するように DirEntry 及び FileEntry構造である。

【００４６】図２は、図１を表す単に１つの考え得るリンクされたリスト配列を表すに過ぎない。この配列は、もしファイルが追加されたが次いで削除されたか、またはディレクトリ名が変更されていれば異なる配列になる。図３は別の考え得る配列を示す。図３は、図１と同一のディレクトリ階層を表しているが、一時期存在したディレクトリ「ビル（BILL）」１１３は削除されている。 FEProm 装置は一回限り書込み可能である（消去された場合を除く）から、本発明の好ましい実施例では図３に示すように、ディレクトリレコード「ビル」１１３をリンクされたリストから物理的に除去しない。ディレクトリまたはファイルは、ディレクトリまたはファイルエントリのステータスをセットすることによってリンクされたリストから削除される。もしディレクトリまたはファイルがコンピュータディスク上に配置されていたのであれば、ディレクトリレコード「ビル」１１３は、ディレクトリレコード「メリー」１０８を指し示しているポインタ１３１をディレクトリレコード「デービッド」１０７内に再書込みすることによって物理的に除去することができる。

【００４７】好ましい実施例は、あるファイルを構成する範囲（extent）をリンクするためにもリンクされたリストデータ構造を使用する。各ファイルはそれに対応付けられたファイルレコードを有し、このファイルレコードは他のデータと共にファイル名を含み、また上述したようにディレクトリ階層内にリンクされている。ある範囲は、そのファイルに関するデータを含むメモリの連続域である。各ファイルは、ファイルデータを含む１またはそれ以上の範囲である。各範囲はそれに対応付けられた範囲レコードを有している。範囲レコードは、他のデータと共にその範囲を指し示すポインタと範囲の長さとを含む。図４はファイル"＼A＼D.DAT"２０２の範囲を示す。範囲レコードＲ１ ２０３、Ｒ２ ２０４及びＲ３ ２０５はリンクされ、対応範囲Ｅ１ ２１１、Ｅ２ ２１２及びＥ３ ２１３を指し示すポインタを含んでいる。ファイルは範囲Ｅ１ ２１１、Ｅ２ ２１２及びＥ３ ２１３の論理的連結である。好ましい実施例では、範囲レコードは後述するように FileInfo 構造である。

【００４８】図４は、ファイル"＼A＼D.DAT"２０２のための単なる１つの考え得るリンクされたリスト配列を表すに過ぎない。図５に同一ファイルを表す別の配列を示す。範囲Ｅ４ ２１４はファイルに追加されたが、削除されている。好ましい実施例では範囲レコードＲ４ ２０６は、ファイルを構成する範囲のリンクされたリストから物理的に除去されていない。そうではなく、レコードの削除を指示するようにレコードのステータスをセットすることによって範囲レコードＲ４ ２０６は論理的に除去されているのである。

【００４９】表Ｃ及びＤは、本発明の好ましい実施例に使用される幾つかのデータ構造を含んでいる。表Ｃに示されているデータ構造は BootRecord である。 BootRecord は、以下に説明するように、ファイルシステムの識別に関するある一般的な情報、 FEProm をアクセスすることができるファイルシステムのバージョン番号、ルートディレクトリを指し示すポインタ、及び付加的なデータを含む。表Ｄに示す第１及び第２の構造は、 DirEntry 構造及び FileEntry構造である。これらの構造の１つが各ディレクトリ及びファイル毎に割当てられる。これらの構造は同一である。変数 SiblingPtr はディレクトリ階層の同一レベルの DirEntry 構造及びFileEntry 構造のリンクされたリスト内の次の同胞を指し示す。変数 PrimaryPtr 及び SecondaryPtr は以下に詳述する。第３の構造は、 FileInfo 構造である。各ファイル範囲は関連 FileInfo 構造を有している。変数 PrimaryPtr は

31

ファイルの FileInfo 構造を指し示す。

表 C

データ構造

```
struct BootRecord
 {  word        Signature;
    dword       SerialNumber;
    word        FPSWriteVersion;
    word        FPSReadVersion;
    word        TotalBlockCount;
    word        SpareBlockCount;
    dword       BlockLen;
    dword       RootDirectoryPtr;
    word        Status;
    byte        VolumeLabelLen;
    word        BootCodeLen;
    byte        VolumeLabel [ ];
    byte        BootCode [ ];
 }
```

定義

Signature	媒体がこのファイルシステムを支援することを指示する値
SerialNumber	VolumeLabel との組合せは特定 FEFrom の独特な識別子である
FPSWriteVersion	このボリュームへ書込むために要求されるファイルシステムの高バイトにおけるバージョン番号及び低バイトにおける改定番号
FPSReadVersion	このボリュームを読出すために要求されるファイルシステムの最早バージョンの高バイトにおけるバージョン番号及び低バイトにおける改定番号
TotalBlockCount	FEFrom内の予備ブロックを含むブロックの合計数
SpareBlockCount	ブロック再利用及びエラー回復に使用可能なブロックの数
BlockLen	バイトで表したブロックの長さ
RootDirectoryPtr	ルートディレクトリを指し示すポインタ
Status	ファイル名フォーマットを指定するデータ
VolumeLabelLen	ボリュームラベル内の文字の数
BootCodeLen	ブートコードアレイ内のバイトの数。もし0ならば媒体はブート不能
VolumeLabel [ ]	ボリュームラベル
BootCode [ ]	オペレーティングシステムに関するブートコード

32

表 D

データ構造

```
struct DirEntry
 {
    word        Status;
    dword       SiblingPtr;
    dword       PrimaryPtr;
    dword       SecondaryPtr;
    byte        Attributes;
    word        Time;
    word        Date;
    byte        NameLen;
    byte        Name [8];
    byte        Ext [3];
}
struct FileEntry
 {
    word        Status;
    dword       SiblingPtr;
    dword       PrimaryPtr;
    dword       SecondaryPtr;
    byte        Attributes;
    word        Time;
    word        Date;
    byte        NameLen;
    byte        Name [8];
    byte        Ext [3];
}
struct FileInfo
 {
    word        Status;
    dword       ExtentPtr;
    dword       PrimaryPtr;
    dword       SecondaryPtr;
    byte        Attributes;
    word        Time;
    word        Date;
    word        ExtentLen;
    word        UncompresedExtentLen;
 }
```

定義

Name	ディレクトリ／ファイル名
Ext	ファイル拡張
Status	
bit #	
0	1：レコードはディレクトリ構造内に存在（存在） 0：レコードはディレクトリ構造から削除済（削除済）
1	1：レコードは現属性、日付、及び時間データを含む（ ATDRecent ）

０：レコードは置換されたデータを含むか、またはデータを含まない(ATDSuperseded)
3－2 １１：未定義
１０：FileInfo
０１：FileEntry
００：DirEntry
4 １： PrimaryPtr は有効ではない
０： PrimaryPtr は有効 (PrimaryPtrValid)
5 １： SecondaryPtr は有効ではない
０： SecondaryPtr は有効 (SecondaryPtrValid)
6 １： SiblingPtr/ExtentPtr は有効ではない
０： StblingPtr/ExtentPtr は有効(SiblingPtrValid/ExtentPtrValid)
DirEntry
15－7 保留
FileEntry
7 １：ファイルは圧縮されていない
０：ファイルは圧縮されている
15－8 圧縮アルゴリズムの識別
FileInfo
7 １：ファイルは圧縮されていない
０：ファイルは圧縮されている
8 １：範囲は圧縮されたブロックの最初のセグメントを含まない
０：範囲は圧縮されたブロックの最初のセグメントを含む
9 １：範囲は圧縮されたブロックの最後のセグメントを含まない
０：範囲は圧縮されたブロックの最後のセグメントを含む
15－10 保留
SiblingPtr 同胞連鎖内の次の DirEntry または FileEntryを指すポインタ
ExtentPtr FileInfoEntry に対応付けられた範囲を指すポインタ
PrimaryPtr DirEntry：ディレクトリ階層内の次に低いレベルの最初の DirEntry または FileEntryを指す
FileEntry ：ファイルに対応付けられた FileInfo エントリのリンクされたリストを指す
FileInfo：ファイルのための次の FileInfo エントリを指す
SecondaryPtr DirEntry：ディレクトリのための次の DirEntry エントリを指す；SecondaryPtrを除くこのエントリの全内容は有効ではなく、無視される
FileEntry ：ファイルのための次の FileEntryエントリを指す；SecondaryPtrを除くこのエントリの全内容は有効ではなく、無視される
FileInfo：ファイルのための次の FileInfo エントリを

指す
Attributes 読出し専用、読出し／書込み等のようなファイル属性
Time 創成または変更の時刻
Date 創成または変更の日付
NameLen バイトで表した名前及び拡張の長さ
Name〔8〕 名前
Ext〔3〕 拡張
ExtentLen バイトで表した範囲の長さ

本発明のファイルシステムは、ディレクトリ及びファイル構造をブロック消去可能な FEProm 内のブロック境界を横切って分散させる。ファイルシステムは、FEProm内の記憶装置の割当て及び割当て解除に FEProm 管理者を使用する。ファイルシステムは、上述したように、リンクされたリスト内のポインタとしてハンドルを使用する。以下の説明では「ハンドル」及び「ポインタ」を互換可能に使用する。図３０は、図２のディレクトリ階層の一部のブロック割当て例を示す。図３０に示した部分はディレクトリ「ルート」、ディレクトリ「 DOS」、ディレクトリ「ワード」、ファイル「 AUTOEXEC.BAT 」、及びファイル「 COMMAND.COM」のための DirEntry 及び FileEntryレコードからなっている。ブロック０はディレクトリ「 DOS」及びディレクトリ「ワード」を含み、ブロック 12 はディレクトリ「ルート」及びファイル「 COMMAND.COM」を含み、そしてブロック 14 はファイル「 AUTOEXEC.BAT 」を含んでいる。

【００５０】図２８は、図３０の例のブロック割当て構造及び領域例を示す。図２８はブロック０ ２４０１、ブロック１２ ２４０２、及びブロック１４ ２４０３を示す。ブロック０ ２４０１は、領域２４２０及び２４３０を含む。領域２４２０はディレクトリ「 DOS」のための DirEntry を含み、領域２４３０はディレクトリ「ワード」のための DirEntry を含む。ブロック０ ２４０１は、見出し２４０４、領域２４２０に対応する Alloc〔0〕エントリ２４２１、及び領域２４３０に対応する Alloc〔1〕エントリ２４３１をも含む。ブロック１２ ２４０２は領域２４１０及び２４５０を含む。領域２４１０はディレクトリ「ルート」のための DirEntry を含み、領域２４５０はファイル「 COMMAND.COM」のための FileEntryを含む。ブロック１２ ２４０２は、ブロック見出し２４０５、領域２４１０に対応する Alloc〔0〕エントリ２４１１、及び領域２４５０に対応するAlloc〔1〕エントリ２４５１をも含む。ブロック１４ ２４０３は、領域２４９０及び２４４０を含む。領域２４９０はブートレコードを含み、領域２４４０はファイル「 AUTOEXEC.BAT 」のための FileEntryを含む。ブロック１４ ２４０３は、ブロック見出し２４０６、領域２４９０に対応する Alloc〔0〕エントリ２４９１、及び領域２４４０に対応する Alloc〔1〕エントリ

２４４１をも含む。

【００５１】図２８では、ポインタ２４０７、２４１３、２４２３、２４３３、２４４３、２４５３、及び２４９３は、ポインタ２４０７から始まりブロック見出し２４０６内のブートレコードまでのディレクトリ階層を限定する。ブートレコード２４９０はブロック１４２４０３内にある。ブロック１４２４０３のための可変ステータスは、このブロックがブートディレクトリを含んでいることを指示する。BootRecordPtr２４０７はブートレコードのための Alloc〔0〕エントリ２４９１を指し示している。Alloc〔0〕エントリ２４９１は、領域２４９０のオフセットを含む変数「オフセット」２４９２を含む。領域２４９０はブートレコードを含む。ブートレコードはディレクトリ「ルート」に対応する Alloc〔0〕エントリ２４１１を指し示すポインタ RootDirectoryPtr ２４９３を含む。Alloc〔0〕エントリ２４１１は、領域２４１０のオフセットを含む変数「オフセット」２４１２を含む。領域２４１０はディレクトリ「ルート」のための DirEntry を含む。ディレクトリ「ルート」の PrimaryPtr ２４１３はディレクトリ「DOS」に対応する Alloc〔0〕エントリ２４２１を指し示す。Alloc〔0〕エントリ２４２１は、領域２４２０のオフセットを含む変数「オフセット」２４２２を含む。【００５２】領域２４２０はディレクトリ DOSのための DirEntry を含む。ディレクトリ「DOS」の SiblingPtr ２４２３はディレクトリ「ワード」のための Alloc〔1〕エントリ２４３１を指し示す。Alloc〔1〕エントリ２４３１は、領域２４３０のオフセットを含む変数「オフセット」２４３２を含む。領域２４３０はディレクトリ「ワード」のための DirEntry を含む。ディレクトリ「ワード」の SiblingPtr ２４３３はファイル「AUTOEXEC.BAT」のための Alloc〔1〕エントリ２４４１を指し示す。Alloc〔1〕エントリ２４４１は、領域２４４０のオフセットを含む変数「オフセット」２４４２を含む。領域２４４０はファイル「AUTOEXEC.BAT」のための FileEntryを含む。ファイル「AUTOEXEC.BAT」のためのポインタ SiblingPtr ２４４３はファイル「COMMAND.COM」のための Alloc〔1〕エントリ２４５１を指し示す。Alloc〔1〕エントリ２４５１は、領域２４５０のオフセットを含む変数「オフセット」２４５２を含む。領域２４５０はファイル「COMMAND.COM」のための FileEntryを含む。SiblingPtr ２４５３は PNULLにセットされ、リンクされたリストの終わりを指示する。

【００５３】このファイルシステムは、ディレクトリの追加及び削除、及びファイルの創成、拡張及び変更を可能にする。図７は、ディレクトリを FEProm へ追加するルーチンの流れ図である。このルーチンの入力パラメタは、新しいディレクトリの完全パスネームと、新しいディレクトリのための属性データである。このルーチンは、親ディレクトリのための DirEntry のアドレスを含むように変数Ｐをセットし、また子ディレクトリのための DirEntry のアドレスを含むように変数Ｃをセットする。例えば、パスネーム "＼Ｐ＼Ｃ" は、ルートディレクトリのサブディレクトリであるＰのサブディレクトリであるディレクトリＣが創成されることを意味する。図８はディレクトリＣがＰの最初のサブディレクトリである場合を示し、図９はディレクトリＣがＰの最初のサブディレクトリではない場合を示す。図８及び９において、実線はディレクトリＣを追加する前のディレクトリ構造を示し、点線はディレクトリＣが追加された後のディレクトリ構造を示す。

【００５４】図７のブロック４０１においてシステムは、ルートディレクトリからの経路を追跡することによってディレクトリＰを探知し、ディレクトリＰのための DirEntry を指し示すように変数Ｐをセットする。ディレクトリＰを探知する時システムは、変数 SecondaryPtr によって置換されていない限り変数 PrimaryPtr を追跡する。ブロック４０２では、システムはディレクトリＣのための DirEntry に領域を割当てる。システムは、FEProm 管理者の手順を呼出すことによって領域を割当てる。システムは、割当て許領域を指し示すように変数Ｃをセットする。以下の説明では FEProm 管理者から戻されるハンドルを領域を指し示すポインタと呼ぶ。ブロック４０３においてシステムは、変数「名、時刻、日付、及び属性」を新たに割当てられたレコード内にセットし、新たに割当てられたエントリがディレクトリエントリであることを指示するように変数「ステータス」をセットする。

【００５５】ブロック４０５乃至４１２においてシステムは、新しいディレクトリエントリを古いディレクトリ構造内へリンクする。システムは、ブロック４０６乃至４１０において新しいディレクトリがＰの最初のサブディレクトリではない場合の状況を処理し、ブロック４１１及び４１２において新しいディレクトリがＰの最初のサブディレクトリである場合の状況を処理する。ブロック４０５において、もし PrimaryPtr が有効であることをＰ－＞「ステータス」が指示すれば、ディレクトリＰはサブディレクトリを有しているか、または以前していたのであり、システムはブロック４０６へ進み、そうでない場合ディレクトリＰはサブディレクトリを有していたことはなく、システムはブロック４１１へ進む。ブロック４１１では、システムは新たに割当てられたディレクトリエントリであるディレクトリＣを指し示すようにＰ－＞ PrimaryPtr をセットして新しいディレクトリへのリンキングを遂行する。ブロック４１２においてシステムは、変数 PrimaryPtr が有効であることを指示するようにＰ－＞「ステータス」をセットしてルーチンを終了する。

【００５６】ブロック４０６においてシステムは、変数 next-ptr をＰ－＞ PrimaryPtr に等しくセットする。

変数 next-ptr は同胞サブディレクトリのリンクされたリスト内の次のディレクトリを指し示すポインタを含む。ブロック４０７において、SiblingPtrが有効であることを next-ptr によって指し示されるレコードのステータスが指示していればシステムはブロック４０８へ進み、そうでなければ同胞のリンクされたリストの終わりに到達したのであり、システムはブロック４０９へ進む。ブロック４０８では、 next-ptr は next-ptr によって指し示されるレコードの SiblingPtr に等しくセットされ、これはリンクされたリスト内の次のディレクトリを指し示すように next-ptr を前進させ、システムはブロック４０７へ進められてリンクされたリストの終わりに達したか否かが決定される。同胞のリンクされたリストの終わりを探索する場合、システムは変数 SiblingPtr を追跡する。システムはブロック４０９において、 next-ptr によって指し示されるレコードの SiblingPtr を、ディレクトリＣのためのDirEntryを指し示すポインタに等しくセットする。ブロック４１０においてシステムは、新たに割当てられたディレクトリエントリを指し示すエントリ内の SiblingPtr が有効であることを指示するように、 next-ptr によって指し示されるレコードの「ステータス」をセットし、ルーチンを終了する。

【００５７】図１０は、新しいファイルに関して FileEntryレコードをファイルシステム内へ追加するルーチンの流れ図である。 FileEntryレコードは階層的な木構造のファイルシステムの単なる葉ノードに過ぎず、 FileEntryレコードを追加するルーチンは図７に示したディレクトリ追加ルーチンに酷似している。主要な相違点はブロック８０３において、レコードがファイルであることを指示するように変数「ステータス」がセットされることである。

【００５８】図１１及び１２は、データをファイルの終わりに追加するためのルーチンの流れ図である。このルーチンには、完全パスネーム、書込まれるデータ及び書込まれるバイトの数が渡される。図１３は拡張されるファイル\L.DAT を含むディレクトリ構造のレイアウトの見本である。実線はファイルが拡張される前の構造を示し、点線はファイルが拡張された後の構造を示す。ファイル「 L.DAT」は始めに FileEntryレコード１１０１、 FileInfo レコード１１０２、及びそれに対応付けられた範囲１１０３を有している。点線は、範囲Ｄ２ １１０５内のファイルへ追加されるデータを有する FileInfo レコード１１０４を表す。

【００５９】図１１のブロック１００１においてシステムは、領域を FEProm 内の新 FileInfo レコードに割当て、そのレコードを指し示すように変数 FI をセットする。システムはブロック１００２において領域をデータ範囲に割当て、その範囲を指し示すように変数Ｄをセットする。システムはブロック１００３において、割当てられたブロックへデータを書込む。ブロック１００４において

システムは割当てられた FileInfo エントリ内に変数「属性、時刻、及び日付」をセットする。ブロック１００５においてシステムは、 FI －> ExtentPtrを割当てられた範囲のハンドルにセットする。ブロック１００５Ａでは、 FI －> ExtentLenが範囲の長さを含むようにセットされる。ブロック１００５Ｂでは FI －>「ステータス」が、「存在」、 ATDRecent、 FileInfo 、及び ExtentPtrvalid にセットされる。ブロック１００６ではシステムは、拡張すべきファイルの FileEntryレコードを探知し、そのレコードのアドレスに FE をセットする。好ましい実施例では、システムは新範囲及び FileInfo レコードを割当てる前に FileEntryレコードを探知して、何等かの割当てが行われる前にファイルが存在するようにしている。

【００６０】図１１の続きである図１２のブロック１００７乃至１０１２においてシステムは、拡張すべきファイルの最後の FileInfo レコード（もし１つが存在すれば）を探知する。システムは FileEntryレコード及び FileInfo レコードの PrimaryPtr または SecondaryPtr を追跡する。有効 SecondaryPtr は、 PrimaryPtr によって指し示されるレコードが、 SecondaryPtr によって指し示されるレコード内のデータによって置換されたことを指示する。ブロック１００７においてシステムは、ポインタ next-ptr を FileEntryレコードを指し示すポインタに等しくセットする。ブロック１００８Ａでは、ポインタ prev-ptr が next-ptr に等しくセットされる。ファイル内の最後の FileInfo レコードが探知されると、ポインタ prev-ptr はそのレコードを指し示すポインタを含む。ブロック１００９では、もし Secondary Ptr が有効であることを next-ptr によって指し示されるレコードのステータスが指示していれば、 PrimaryPtr によって指し示されるレコード内のデータは攪拌されたのであり、システムはブロック１０１１へ進み、そうでない場合はシステムはブロック１０１０へ進む。ブロック１０１０においてシステムは next-ptr を、 next-ptr によって指し示されるレコードの PrimaryPtr に等しくセットしてリンクされたリスト内の次のレコードを指し示すポインタを入手し、ブロック１０１２へ進む。ブロック１０１１ではシステムは next-ptr を、 next-ptr によって指し示されるレコードの SecondaryPtr に等しくセットしてリンクされたリスト内の次のレコードを指し示すポインタを入手し、ブロック１００８Ａへ進む。ブロック１０１２では、もし next-ptr が有効であれば、リンクされたリストの終わりに達したのであり、システムはブロック１０１３へ進み、そうでない場合にはシステムは１００８Ａへ戻ってリンクされたリスト内の次のレコードを処理する。ブロック１０１３ではシステムは prev-ptr によって指し示されるレコードの PrimaryPtr を、 FI を指し示すポインタに等しくセットしてファイルの拡張を遂行する。ブロック１０１４にお

てシステムはprev-ptrによって指し示されるレコードの「ステータス」を PrimaryPtrValidに等しくセットしてルーチンを終了する。

【００６１】図１４及び１５は、ファイル内のデータを更新する「ファイル更新」ルーチンの流れ図である。このルーチンのパラメタは、変更された関連範囲を持つためのFileInfoブロックのアドレスであるR上、新データのための範囲内への オフセットである extent-offset と、新データである new-data と、新データの長さである data-lengthとである。少なくともあるブロックが消去されるまでは、PEPromは１回書込み装置であるから、データが記憶される領域は、ファイルへの更新が発生する時には再書込みはできない。以下に説明するように好ましい実施例では更新されるデータは PEProm の異なる領域へ書込まれる。

【００６２】図１６は、あるファイルのための FileInfo レコードのリンクされたリストの典型的部分を示す。ファイル更新ルーチンは陰影を施した領域１３０１によって返されるデータを置換する。図１７は変更されるデータが PEProm へ書込まれた後のリンクされたリストの構造を示す。３つの FileInfo レコードR１ １４１１、R２ １４１２、及びR３ １４１３がリンクされたリスト内へ挿入されている。範囲全体が再書込みされるのではなく、実際に変更された部分だけが再書込みされるのである。ルーチンは範囲を３つの区分D１ １４０１、D２ １４０２、及びD３ １４０３に分割する。区分D１ １４０１及びD３ １４０３は、更新によって変化しないデータを含み、区分D２ １４０２は変化するデータを含む。各区分は対応する FileInfo レコードを有する。 FileInfo レコードR１ １４１１、R２ １４１２、及びR３ １４１３はそれらの PrimaryPtrフィールドを通してリンクされている。またR１ １４１１及びR３ １４１３内の ExtentPtrフィールドはそれらの対応範囲区分を指し示すようにセットされ、 ExtentLenフィールドがセットされる。新範囲は、レコードR２ １４１２によって指し示される区分である新D２ １４０４に対応する新データに割当てられる。レコードR １４１０の SecondaryPtr は FileInfo R１ １４１１を指し示し、R １４１０の PrimaryPtr が置換されたことを指示する。 FileInfoレコードR３ １４１３の PrimaryPtr は FileInfo レコードR１４１０の PrimaryPtr 内に含まれている値にセットされてリンクを完了する。ファイル更新ルーチンは、ブロック内に３つの新 Allocエントリを追加するための範囲を含む十分なスペースが存在することに依存する。これら３つの Allocエントリは、１つの領域ではなく３つの領域内に範囲を再定義する。もし十分なスペースが存在しなければ、ブロックの再利用によって空きスペースを発生させることができ、ファイル更新ルーチンを呼出すことができる。しかし、もし十分な空きスペースが存

在しなければ、その範囲内のデータは新範囲へ移動させられる。もしデータが移動させられれば、新データは古データと統合され、１つだけの FileInfo レコードを有する新ブロック内の１領域へ書込まれる。古ブロック内の領域は割当て解除される。好ましい実施例では、 FEProm 管理者は既存領域の部分を指し示すAlloc エントリの追加を支援する。図１７の例は、ブロックへ追加する３つの新Alloc エントリを必要とし、これらはD１、D２、及びD３に関連付けられた新たに定義された領域に対応する。D２のための Allocエントリは割当て解除され、D１及びD３のための Allocエントリが割当てられる。区分D１、D２、及びD３からなる領域に対応する古い Allocエントリのステータスは、それが置換されたことを指示するようにセットされる。置換済のステータスは、 Allocエントリが本質的に対応領域を用いずに割当て解除されることを指示する。

【００６３】図１４のブロック１２０１においてシステムは FileInfo レコードのための３つの領域を割当て、領域のアドレスを含むように変数R１、R２、及びR３をセットする。ブロック１２０２においてもしR－>「ステータス」が ATDRecentを指示していれば、システムはR１－>「時刻」、R１－>「日付」、及びR１－>「属性」をR内の値にセットし、R１－>「ステータス」を ATDRecentにセットし、そうではない場合には、システムはこれらのフィールドを FNULLにしたままである。好ましい実施例では、 FEProm 管理者は Allocエントリを置換にセットするのと、 Allocエントリを既存領域に割当てるのを支援する。ブロック１２０３においてシステムはある領域を新データに割当て、 R2NewDataをその領域のアドレスにセットする。ブロック１２０４では、３つの Allocエントリが割当てられる。これらのエントリはD１、D２、及びD３を指し示すように初期化される。D１、D２、及びD３からなる領域を指し示す Allocエントリのステータスは置換にセットされる。ブロック１２０５においてシステムは R2NewDataによってアドレスされた新領域へ new-data を書込む。システムはブロック１２０６乃至１２０８Aにおいてデータを FileInfo レコードR２内にセットする。ブロック１２０６においてシステムは、R２－> ExtentPtrを新データのための領域を指し示すポインタにに等しくセットする。ブロック１２０７では、R２－> ExtentLenを新領域の長さに等しくセットする。ブロック１２０８においては、R２－> PrimaryPtr がR３を指し示すポインタにセットされる。ブロック１２０８Aにおいてシステムは、ExtentPtr 及び PrimaryPtr が有効であることを指示するようにR２－>「ステータス」をセットする。

【００６４】図１４の続きである図１５のブロック１２０９乃至１２１１Aにおいてシステムは、データを FileInfo レコードR３内にセットする。ブロック１２０９ではD３領域を指し示すポインタに等しくR３－> Ext

entPtrをセットする。ブロック１２１０ではＲ３－＞ＥｘｔｅｎｔＬｅｎをＤ３領域の長さに等しくセットする。ブロック１２１１においてシステムは、Ｒ３－＞ＰｒｉｍａｒｙＰｔｒをＲ－＞ＰｒｉｍａｒｙＰｔｒに等しくセットする。ブロック１２１１Ａでは、ＥｘｔｅｎｔＰｔｒ及びＰｒｉｍａｒｙＰｔｒが有効であることを指示するようにＲ３－＞「ステータス」をセットする。

【００６５】ブロック１２１２乃至１２１４Ａにおいてシステムは、データをＦｉｌｅＩｎｆｏレコードＲ１内にセットする。ブロック１２１２ではＤ１領域を指し示すポインタに等しくＲ１－＞ＥｘｔｅｎｔＰｔｒをセットする。ブロック１２１３ではＲ１－＞ＥｘｔｅｎｔＬｅｎをＤ３領域の長さに等しくセットする。ブロック１２１４においてシステムは、Ｒ１－＞ＰｒｉｍａｒｙＰｔｒをＲ２を指し示すポインタにセットする。ブロック１２１４Ａでは、ＥｘｔｅｎｔＰｔｒ及びＰｒｉｍａｒｙＰｔｒが有効であることを指示するようにＲ１－＞「ステータス」をセットする。

【００６６】ブロック１２１５においては、Ｒ１を指し示すポインタに等しくなるようにＲ－＞ＳｅｃｏｎｄａｒｙＰｔｒがセットされる。ブロック１２１６では、ＳｅｃｏｎｄａｒｙＰｔｒが有効であることを指示するようにＲ－＞「ステータス」がセットされる。これでルーチンが終了する。図１８及び１９は、ファイル更新の特別な場合の結合のＦｉｌｅＩｎｆｏリストを示す。これらの特別な場合を処理するためのルーチンは、図１４、１５に示したファイル更新の一般的な場合の処理に必要なルーチンの部分集合である。図１８において、範囲の始まりから始まるデータが更新される。区分Ｄ１　１５０１は更新すべき範囲の始まりのデータを含み、区分Ｄ２　１５０２は更新されない範囲の終わりのデータを含む。２つの新ＦｉｌｅＩｎｆｏレコードだけが必要である。第１のＦｉｌｅＩｎｆｏレコードＲ１　１５１１は新データ１５０３を指し示し、第２のＦｉｌｅＩｎｆｏレコードＲ２　１５１２は区分Ｄ２　１５０２内の古いデータを指し示す。図１９に示すように範囲境界上で終わるデータが更新される場合にも同様な状況が発生する。ファイル更新の一般的な場合のように、Ｄ１及びＤ２を含む古い領域は、古い領域を含むブロックに２つの新しい割当て表エントリを割当てることによって２つの領域に細分される。またもしエントリのために十分なスペースが存在しなければ、変更されないデータは新ブロックへ移動させられる。

【００６７】図２０は更新データが範囲境界にまたがる場合のＦｉｌｅＩｎｆｏレコードのためのリンクされたリストを示す。図２１は、ＦＥＰｒｏｍからディレクトリを削除するルーチンの流れ図である。ファイルを削除するルーチンは、対応付けられたＦｉｌｅＩｎｆｏレコードが割当て解除されること以外は同一である。このルーチンは、ＤｉｒＥｎｔｒｙのステータスを、それが削除されることを指示するようにセットする。ブロック１８０１においてシステムは、削除すべきディレクトリを探知し、そのディレクトリのアドレスを含むように変数「ポインタＤ」をセットする。ブロック１８０２では、そのディレクトリを削除することを指示するようにＤ－＞「ステータス」をセットする。

【００６８】ディレクトリまたはファイルの名前は、新しいＤｉｒＥｎｔｒｙまたはＦｉｌｅＥｎｔｒｙをそれぞれ割当て、次いで新しいエントリを指し示すように古いエントリのＳｅｃｏｎｄａｒｙＰｔｒをセットすることによって変更される。図２３は、名前が“Ｂ.ＤＡＴ”に変更される時の、“Ｄ.ＤＡＴ”のためのファイルエントリを実線で、また変更を点線で示してある。新エントリはＦｉｌｅＩｎｆｏエントリ、ディレクトリ構造、及び古いエントリに対応付けられた範囲のリンクされたリストを指し示す。

【００６９】図２２はファイル名の変更を実現する好ましいサブルーチンの流れ図である。ディレクトリ名を変更するためのサブルーチンは、対応付けられる範囲が存在しないことを除いて同一である。このルーチンへの入力パラメタは、ファイルの同胞及び新ファイル名である。ブロック１９０１においてシステムはディレクトリを通してシステムを探索し、名前を変更すべきファイルを探知し、ＦｉｌｅＥｎｔｒｙを指し示すように変数Ｐをセットする。システムは、そのファイルのためのエントリのリンクされたリスト内の最後の　ＦｉｌｅＥｎｔｒｙを探索する。ファイルは各名前変更毎にエントリを有する。

【００７０】ブロック１９０４においてシステムは、領域を新ＦｉｌｅＥｎｔｒｙに割当て、その領域を指し示すように変数Ｃをセットする。ブロック１９０５においてシステムはＣ－＞「名前」を新ファイル名に等しくセットし、Ｃ－＞「属性」、Ｃ－＞「日付」、Ｃ－＞「時刻」をセットし、ＡＴＤＲｅｃｅｎｔに置換えるファイルエントリに基づいてＣ－＞「ステータス」をセットする。ブロック１９０６においてシステムは、Ｃ－＞ＳｉｂｌｉｎｇＰｔｒをＰ－＞ＳｉｂｌｉｎｇＰｔｒに等しくセットしてエントリをディレクトリ階層にリンクする。ブロック１９０９では、Ｃ－＞ＰｒｉｍａｒｙＰｔｒがＰ－＞ＰｒｉｍａｒｙＰｔｒに等しくセットされ、新エントリは範囲のリストにリンクされる。ブロック１９０９Ａでは、ＥｘｔｅｎｔＰｔｒ及びＰｒｉｍａｒｙＰｔｒが有効であることを指示するためにＣ－＞「ステータス」がセットされる。ブロック１９１０ではシステムはＣを指し示すポインタに等しくＰ－＞ＳｅｃｏｎｄａｒｙＰｔｒをセットする。ブロック１９１１においてシステムはＳｅｃｏｎｄａｒｙＰｔｒが有効であることを指示するようにＰ－＞「ステータス」をセットし、古いエントリの置換を完了させてルーチンを終了させる。

【００７１】図２４及び２５は、あるファイルの属性データを変更するルーチンの流れ図である。あるファイルに関連する属性データは、ＡＴＤＲｅｃｅｎｔのステータスを有し、また「性、日付、及び時刻」フィールドがＮＵＬＬにセットされている第１のＦｉｌｅＩｎｆｏエントリを選択することによって変更される。もしこのようなフィールドが

存在しなければ、新しい FileInfo エントリが創成され、選択される。次いでシステムは「属性、日付、及び時刻」フィールドを選択された FileInfo エントリ内にセットする。先に最新の「属性、日付、及び時刻」データを記憶していたFileInfoエントリは、 ATDSuperseded にセットされているステータスを有している。入力パラメタはパスネーム及び属性データである。ブロック２１０１においてシステムはディレクトリ構造を通して探索してファイルを探知し、 FileEntryを指し示すように変数Ｐをセットする。ブロック２１０２においてもしＰ－>「ステータス」が ATDRecentを指示していれば、 FileEntryは最新の属性データを含んでおり、システムはブロック２１０３に進み、そうでない場合にはシステムはブロック２１０４に進む。ブロック２１０３においてシステムは変数Ｘを変数Ｐにセットして図２５のブロック２１１１に進む。ブロック２１０４ではシステムは変数ＣをＰ－> PrimaryPtr に等しくセットする。システムはブロック２１０５乃至２１０８をループして最新の属性データを指示しているステータスを有するFileInfo エントリを探索する。ブロック２１０５においてもし SecondaryPtr が有効であることをＣ－>「ステータス」が指示していれば、システムはブロック２１０６へ進み、そうでない場合はシステムはブロック２１０７へ進む。ブロック２１０６では変数ＣはＣ－> SecondaryPtr にセットされて置換されるエントリを指し示すようにされ、ブロック２１０５へループバックされる。ブロック２１０７においてもしＣ－>「ステータス」が ATDRecentを指示していればFileInfoエントリは最新の属性データを含んでいるので、システムはブロック２１０９へ進み、そうでない場合にはシステムはブロック２１０８へ進む。ブロック２１０８においてシステムは変数ＣをＣ－> PrimaryPtr に等しくセットし、ブロック２１０５へループバックする。ブロック２１０９においてはシステムは変数Ｘを変数Ｃにセットし、図２５のブロック２１１１へ進む。

【００７２】システムは、ブロック２１１１において変数Ｙを変数Ｘに初期化する。変数Ｘは最新の属性データを有するエントリを指し示す。ブロック２１１２乃至２１１９においては、再新のステータスを有し FNULLにセットされた属性データを有する次のエントリを探知する。ブロック２１１２においてもし PrimaryPtr が有効であることをＹ－>「ステータス」が指示していれば、システムはブロック２１１３へ進み、そうでなければ新エントリを追加するためにブロック２１２０へ進む。ブロック２１１３では、変数ＹはＹ－> PrimaryPtr にセットされる。ブロック２１１４においてもし SecondaryPtr が有効であることをＹ－>「ステータス」が指示していれば、システムはブロック２１１５へ進み、そうでなければブロック２１１６へ進む。ブロック２１１５では変数ＹがＹ－> SecondaryPtr に等しくセットされ、

ブロック２１１４へループバックされる。ブロック２１１６においてもしＹ－>「ステータス」が ATDRecentにセットされていればシステムはブロック２１１７へ進み、そうでなければブロック２１１２へ戻される。ブロック２１１７においてもしＹ－>「属性」、Ｙ－>「日付」、及びＹ－>「時刻」が FNULLに等しければシステムはブロック２１１８へ進み、そうでなければブロック２１１２へループバックされる。ブロック２１１８において、Ｙ－>「属性」、Ｙ－>「日付」及びＹ－>「時刻」は新しい属性データにセットされる。ブロック２１１９ではＸ－>「ステータス」が ATDSupersededに等しくセットされ、ルーチンは完了する。

【００７３】ブロック２１２０乃至２１２３においてシステムは新 FileInfo エントリを割当てて初期化する。ブロック２１２０では新 FileInfo エントリが割当てられ、新エントリを指し示すように変数Ｚがセットされる。ブロック２１２１では、システムはＺ－>「属性」、Ｚ－>「日付」、及びＺ－>「時刻」を新属性データにセットし、Ｚ－>「ステータス」を「存在」、 ATDRecent、及び FileInfo にセットし、Ｚ－>ExtentPtr をＹ－>ExtentPtr にセットし、そしてＺ－>ExtentLen をＹ－>ExtentLen にセットする。ブロック２１２２ではＹ－>SecondaryPtrが変数Ｚに等しくセットされ、 SecondaryPtr が有効であることを指示するようにＹ－>「ステータス」がセットされる。ブロック２１２３では、Ｘ－>「ステータス」が ATDSupersededに等しくセットされ、エントリが最早現属性データを含んでいないことを指示し、ルーチンが完了する。

【００７４】以上に本発明を好ましい実施例に関して説明したが、本発明はこの実施例に限定されるものではない。当業者ならば本発明の思想に基づく多くの変更が明白であろう。本発明の範囲は特許請求の範囲によってのみ限定されるものである。

【図面の簡単な説明】

【図１】ディレクトリ及びファイルのサンプルハイアラーキ構造もしくはツリー構造の編成を示す図である。

【図２】図１の同じディレクトリ構造を示すリンクされたリスト構造体の図である。

【図３】図１の同じディレクトリ構造を表す別のリンクされたリスト構造体を示した図である。

【図４】ファイル名"＼A＼d．DAT"のファイルに対するリンクされたリストの構造を示す図である。

【図５】ファイル名"＼A＼d．DAT"のファイルに対する別のリンクされたリストの構造を示す図である。

【図６】本発明の好ましい実施例によるブロック消去可能なＦＥＰｒｏｍのレイアウトを示す図である。

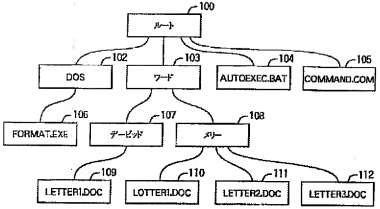
【図７】本発明の好ましい実施例によるＡｄｄ Ｄｉｒｅｃｔｏｒｙルーチンを示す流れ線図である。

【図８】本発明の好ましい実施例によりディレクトリ構造に新たにディレクトリが追加される前後の図である。

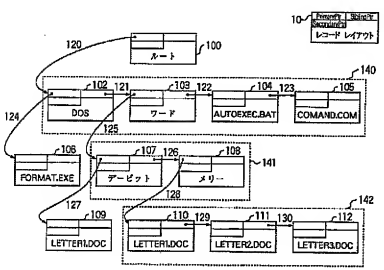
【図９】本発明の好ましい実施例によりディレクトリ構造に新たにディレクトリが追加される前後の図である。
【図１０】本発明の好ましい実施例によるＡｄｄ＿Ｆｉｌｅルーチンを示す流れ線図である。
【図１１】本発明の好ましい実施例によるＥｘｔｅｎｄ＿Ｆｉｌｅルーチンを示す流れ線図である。
【図１２】本発明の好ましい実施例によるＥｘｔｅｎｄ＿Ｆｉｌｅルーチンを示す流れ線図である。
【図１３】本発明の好ましい実施例を用いたサンプルディレクトリ及びファイルレイアウトを示す図である。
【図１４】本発明の好ましい実施例によるＵｐｄａｔｅ＿Ｆｉｌｅルーチンを示す流れ線図である。
【図１５】本発明の好ましい実施例によるＵｐｄａｔｅ＿Ｆｉｌｅルーチンを示す流れ線図である。
【図１６】本発明の好ましい実施例により更新される前後のファイルのサンプル部分を示す図である。
【図１７】本発明の好ましい実施例により更新される前後のファイルのサンプル部分を示す図である。
【図１８】本発明の好ましい実施例によりファイルが更新された後のファイルのサンプル部分を示す図である。
【図１９】本発明の好ましい実施例によりファイルが更新された後のファイルのサンプル部分を示す図である。
【図２０】本発明の好ましい実施例によりファイルが更新された後のファイルのサンプル部分を示す図である。
【図２１】本発明の好ましい実施例によるＤｅｌ＿Ｄｉｒｅｃｔｏｒｙルーチンを示す流れ線図である。
【図２２】本発明の好ましい実施例によるＣｈａｎｇｅ＿Ｆｉｌｅ Ｎａｍｅルーチンを示す流れ線図である。
【図２３】名前を変化したファイルに対するディレクトリ構造の前後を表す図である。
【図２４】本発明の好ましい実施例によるＣｈａｎｇｅ＿Ａｔｔｒｉｂｕｔｅ＿Ｄａｔａルーチンを示す流れ線図である。
【図２５】本発明の好ましい実施例によるＣｈａｎｇｅ＿Ａｔｔｒｉｂｕｔｅ Ｄａｔａルーチンを示す流れ線図である。
【図２６】本発明の好ましい実施例におけるブロックのレイアウトを示す図である。
【図２７】好ましい実施例においてリクレイミングを行った後の図２６のブロックのレイアウトを示す図である。

【図２８】図３０のサンプルに対するサンプルブロック割り当て構造及び領域を示す図である。
【図２９】ハンドルの参照解除を示す図である。
【図３０】図２のディレクトリハイアラーキの一部分に対するサンプルブロック割り当てを示す図である。
【図３１】好ましい実施例における初期化プロセスを示す流れ線図である。
【図３２】好ましい実施例におけるブロック割り当てルーチンを示す流れ線図である。
【図３３】好ましい実施例における領域割り当てルーチンを示す流れ線図である。
【図３４】好ましい実施例におけるブロックリクレイミングルーチンを示す流れ線図である。
【図３５】Ａｌｌｏｃエントリのステータスを示す状態図である。
【図３６】ブロックのステータスを示す状態図である。
【符号の説明】
１００　ディレクトリ「ルート」
１０２　「ＤＯＳ」サブディレクトリ
１０３　「ワード」サブディレクトリ
１０４　「ＡＵＴＯＥＸＥＣ．ＢＡＴ」ファイル
１０５　「ＣＯＭＭＡＮＤ．ＣＯＭ」ファイル
１０６　「ＦＯＲＭＡＴＥＸＥ」ファイル
１０７　「デーピッド」サブディレクトリ
１０８　「メリー」サブディレクトリ
１０９　「ＬＥＴＴＥＲ１．ＤＯＣ」ファイル
１１０　「ＬＥＴＴＥＲ１．ＤＯＣ」ファイル
１１１　「ＬＥＴＴＥＲ２．ＤＯＣ」ファイル
１１２　「ＬＥＴＴＥＲ３．ＤＯＣ」ファイル
１１３　「ビル」ディレクトリ
１２０－１３２　ポインタ
１４０－１４２　リンクされたリスト
２００－２０２　ファイル
２０３－２０６　範囲レコード
２１１－２１４　範囲
３０１　ＦＥＰｒｏｍ
３０２　ブロック
２３０２　ブロック割り当て構造体
２３０３　データ領域
２３０４　自由スペース
２３１０－２３１５　オフセット

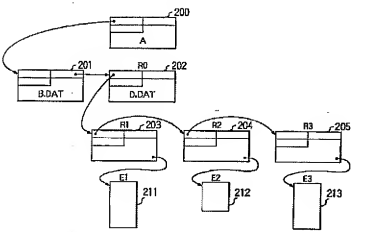
【図１】



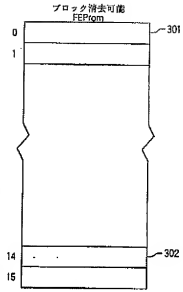
【図２】



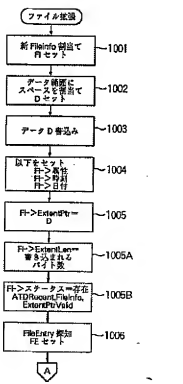
【図４】



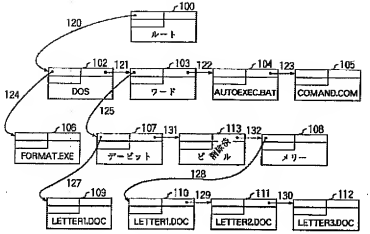
【図６】



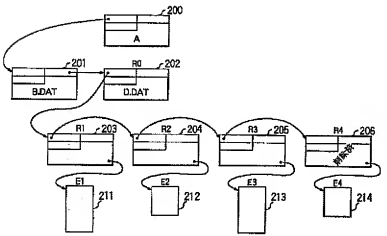
【図１１】



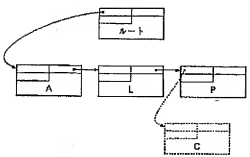
【図３】



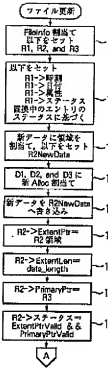
【図５】



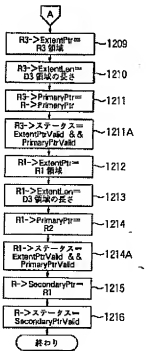
【図８】



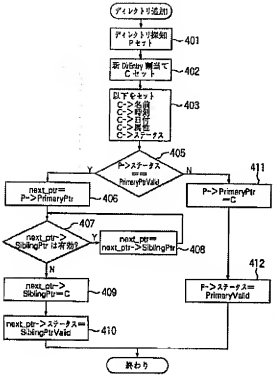
【図１４】



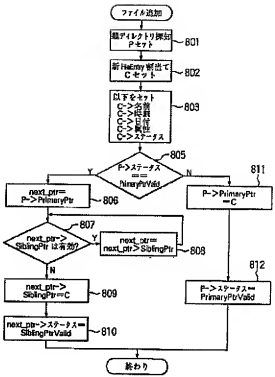
【図１５】



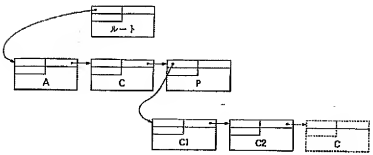
【図７】


ディレクトリ追加
ディレクトリ探知
Pセット
401
新 DirEntry 割当て
Cセット
402
以下をセット
C->名前
C->日付
C->時間
C->属性
C->ステータス
403
405
P->ステータス
==
PrimaryPtrValid
Y
N
next_ptr=
P->PrimaryPtr
406
411
P->PrimaryPtr
=C
407
next_ptr->
SiblingPtr は有効?
Y
next_ptr=
next_ptr->SiblingPtr
408
N
412
next_ptr->
SiblingPtr=C
409
P->ステータス=
PrimaryPtrValid
next_ptr->ステータス=
SiblingPtrValid
410
終わり

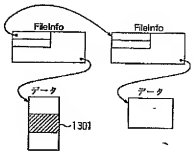
【図１０】


ファイル追加
親ディレクトリ探知
Pセット
801
新 FileEntry 割当て
Cセット
802
以下をセット
C->名前
C->日付
C->時間
C->属性
C->ステータス
803
805
P->ステータス
==
PrimaryPtrValid
Y
N
next_ptr=
P->PrimaryPtr
806
811
P->PrimaryPtr
=C
807
next_ptr->
SiblingPtr は有効?
Y
next_ptr=
next_ptr->SiblingPtr
808
N
812
next_ptr->
SiblingPtr=C
809
P->ステータス=
PrimaryPtrValid
next_ptr->ステータス=
SiblingPtrValid
810
終わり

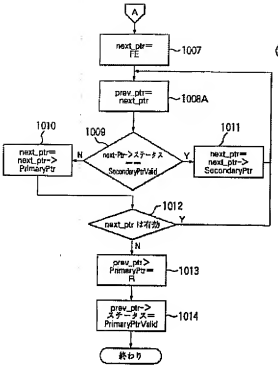
【図９】


ルート
A
C
P
C1
C2
C

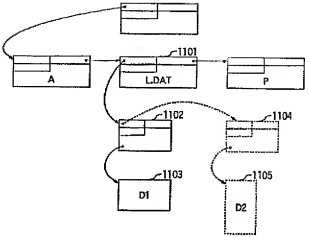
【図１６】


FileInfo
FileInfo
データ
データ
1301

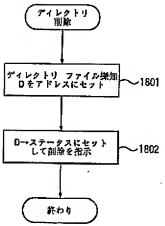
【図１２】



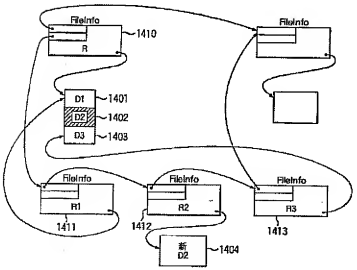
【図１３】



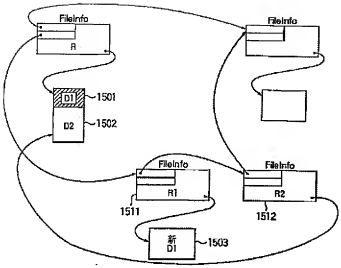
【図２１】



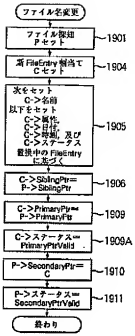
【図１７】



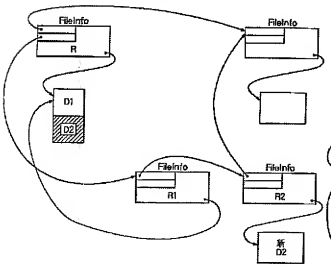
【図１８】



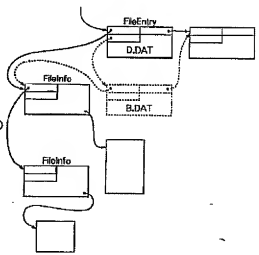
【図２２】



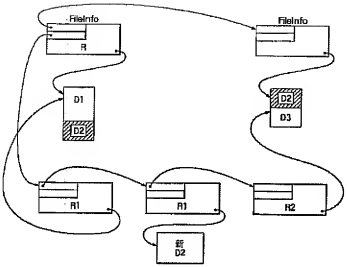
【図１９】



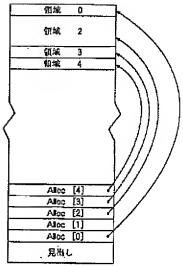
【図２３】



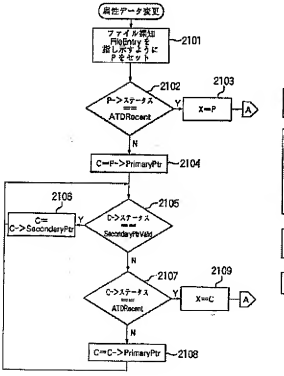
【図２０】



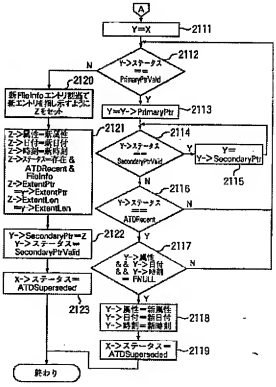
【図２７】



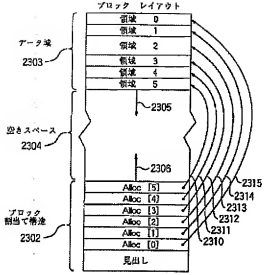
【図２４】



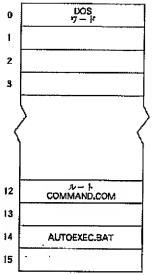
【図２５】



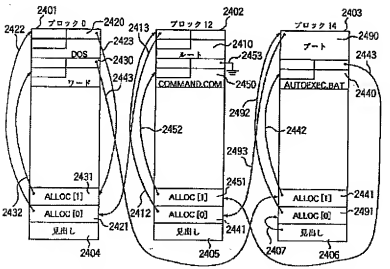
【図２６】


ブロック レイアウト
領域 0
領域 1
領域 2
領域 3
領域 4
領域 5
データ域
2303
2305
空きスペース
2304
2306
Alloc [5]
Alloc [4]
Alloc [3]
Alloc [2]
Alloc [1]
Alloc [0]
見出し
ブロック
割当て構造
2302
2315
2314
2313
2312
2311
2310

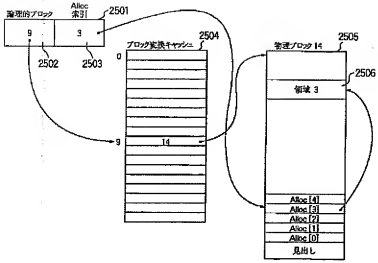
【図３０】


0
1
2
3
12
13
14
15
DOS
ワード
ルート
COMMAND.COM
AUTOEXEC.BAT

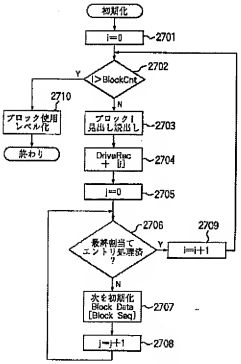
【図２８】


2401
ブロック 0
2420
2422
2423
DOS
2430
ワード
2443
2431
ALLOC [1]
2432
ALLOC [0]
2421
見出し
2404
2413
2402
ブロック 12
2410
ルート
2453
COMMAND.COM
2450
2452
2451
ALLOC [1]
2412
ALLOC [0]
2441
見出し
2405
2403
ブロック 14
2490
ブート
2443
AUTOEXEC.BAT
2440
2492
2442
2493
2441
ALLOC [1]
2491
ALLOC [0]
見出し
2407
2406

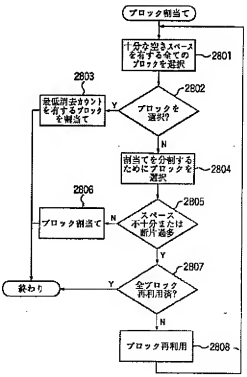
【図２９】


論理的ブロック
Alloc 索引
2501
9
3
2502
2503
ブロック変換キャッシュ
2504
0
9
14
管理ブロック14
2505
領域３
2506
Alloc[4]
Alloc[3]
Alloc[2]
Alloc[1]
Alloc[0]
見出し

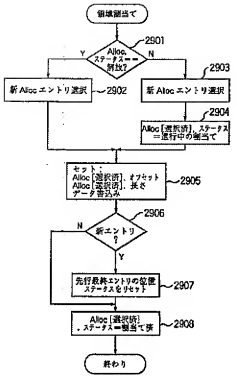
【図３１】


初期化
i=0
2701
i>BlockCnt
2702
Y
N
2710
ブロック使用レベル化
読わり
ブロックi見出し設出し
2703
DriveRec＋[i]
2704
i=0
2705
最終割当てエントリ記憶済?
2706
Y
2709
i=i+1
N
次を初期化 Block Data [Block Seq]
2707
j=j+1
2708

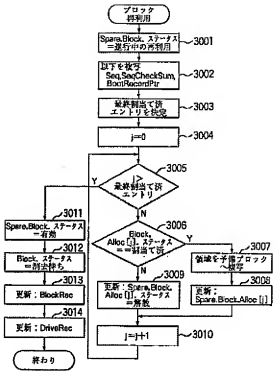
【図３２】


ブロック割当て
十分な空きスペースを有する今でのブロックを選択
2801
2803
最低消去カウントを有するブロックを割当て
2802
ブロックを選択?
Y
N
割当てを分割するためにブロックを選択
2804
2806
ブロック割当て
N
スペース不十分または断片過多
2805
Y
2807
読わり
Y
全ブロック再利用済?
N
ブロック再利用
2808

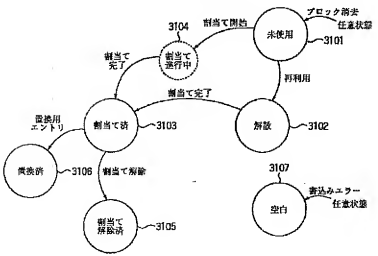
【図３３】


領域割当て
Alloc.ステータス＝解放?
Y
N
2901
新 Alloc エントリ選択
2902
新 Alloc エントリ選択
2903
Alloc［選択済］.ステータス＝実行中の割当て
2904
セット：Alloc［選択済］.オフセット
Alloc［選択済］.長さ
データ書込み
2905
新エントリ
2906
N
Y
先行最終エントリの位置ステータスをリセット
2907
Alloc［選択済］.ステータス＝割当て済
2908
終わり

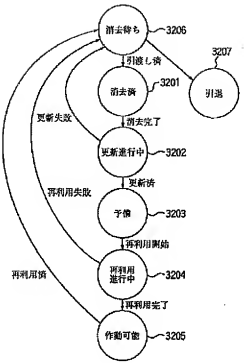
【図３４】


ブロック再利用
Spare.Block.ステータス＝実行中の再利用
3001
以下を設定
Seq.SeqCheckSum.BlockRecordPtr
3002
最終割当て済エントリを記憶
3003
j:=0
3004
最終割当て済エントリ
3005
Y
N
Spare.Block.ステータス＝有効
3011
Block.ステータス＝解放待ち
3012
更新：BlockRec
3013
更新：DriveRec
3014
Block.Alloc［j］.ステータス＝＝割当て済
3006
Y
N
領域を子孫ブロックへ継承
3007
更新：Spare.Block.Alloc［j］.ステータス＝解放
3009
更新：Spare.Block.Alloc［j］
3008
j:=j+1
3010
終わり

【図３５】


割当て開始
3104
ブロック消去
任意状態
未使用
3101
割当て実行中
割当て完了
再利用
割当て完了
割当て済
3103
解放
3102
置換用エントリ
廃棄済
3106
割当て解除
割当て解放済
3105
3107
空白
書込みエラー
任意状態

【図３６】



フロントページの続き

(72)発明者　スリラム　ラジャゴパラン
アメリカ合衆国　ワシントン州　98007
ベルヴィュー　ビー201　ノースイースト
サーティフィフス　14550